

A Thesis Submitted for the Degree of PhD at the University of Warwick

Permanent WRAP URL:

<http://wrap.warwick.ac.uk/93676>

Copyright and reuse:

This thesis is made available online and is protected by original copyright.

Please scroll down to view the document itself.

Please refer to the repository record for this item for information to help you to cite it.

Our policy information is available from the repository home page.

For more information, please contact the WRAP Team at: wrap@warwick.ac.uk



Reasoning about Systems with Evolving Structure

by

Anna Philippou

Thesis

Submitted to the The University of Warwick

for admission to the degree of **Doctor of Philosophy**

Department of Computer Science

December 1996

8030050

Contents

| | |
|--|------|
| Acknowledgments | v |
| Declarations | vi |
| Abstract | vii |
| Glossary | viii |
| Chapter 1 Introduction | 1 |
| 1.1 Mobility | 2 |
| 1.2 Determinacy and Confluence | 4 |
| 1.3 Concurrency Control of <i>B</i> -trees | 6 |
| 1.4 Object-oriented languages | 8 |
| 1.5 Summary of the thesis | 10 |
| Chapter 2 Background | 13 |
| 2.1 The π_v -calculus | 14 |
| 2.2 Sorting | 19 |
| 2.3 Labelled transition semantics | 22 |
| 2.4 Behavioural equivalence | 25 |
| 2.4.1 Branching bisimilarity | 27 |
| 2.4.2 Coupled simulations | 29 |
| 2.4.3 Divergence | 31 |
| 2.5 Friendly agents | 33 |
| Chapter 3 Determinacy and Confluence | 40 |
| 3.1 Determinacy | 41 |
| 3.2 Preserving Determinacy | 43 |
| 3.3 \asymp -determinacy | 45 |
| 3.4 Confluence | 47 |

| | | |
|------------------|--|------------|
| 3.5 | Preserving Confluence | 54 |
| 3.6 | \asymp -confluence | 61 |
| 3.7 | Divergence | 63 |
| 3.8 | Π -confluence | 72 |
| 3.9 | Labelled transition systems | 74 |
| 3.10 | A Protocol Example | 76 |
| 3.10.1 | The Algorithm | 77 |
| 3.10.2 | Correctness Proof | 80 |
| Chapter 4 | Partial Confluence | 91 |
| 4.1 | Partial confluence and divergence | 93 |
| 4.2 | db^R -bisimilarity | 106 |
| 4.3 | Social confluence | 108 |
| 4.4 | When names are not new | 118 |
| Chapter 5 | Concurrent Operations on B-trees | 127 |
| 5.1 | The structure | 129 |
| 5.1.1 | The low key | 136 |
| 5.2 | Correctness Proof | 137 |
| 5.2.1 | Analysis of T_0 | 138 |
| 5.2.2 | Analysis of Q_0 | 141 |
| 5.2.3 | Main Results | 151 |
| 5.2.4 | Full Results | 155 |
| 5.3 | The Concurrent Search Structure | 158 |
| 5.4 | Deletion and compression | 160 |
| 5.4.1 | The Algorithms | 162 |
| 5.4.2 | Correctness | 165 |
| Chapter 6 | Concurrent Objects | 167 |
| 6.1 | The programming language | 169 |
| 6.2 | The semantic definition | 173 |
| 6.3 | Determinacy | 178 |
| 6.4 | A D -program is confluent | 180 |
| 6.5 | Transformations | 190 |
| 6.6 | Correctness | 196 |
| 6.6.1 | A guarded community | 197 |
| 6.6.2 | Deadlock freedom | 201 |
| 6.6.3 | Partial Confluence | 204 |

| | | |
|--|---|------------|
| 6.6.4 | Transformation 1 | 207 |
| 6.6.5 | Transformation 2 | 217 |
| Chapter 7 Conclusions and Future Work | | 224 |
| 7.1 | Conclusions | 224 |
| 7.1.1 | Determinacy and Confluence in the π_v -calculus | 225 |
| 7.1.2 | B -tree operations | 226 |
| 7.1.3 | Determinacy in $\pi o\beta\lambda$ | 227 |
| 7.1.4 | The transformations | 228 |
| 7.2 | Future work | 229 |
| 7.2.1 | Determinacy and Confluence | 229 |
| 7.2.2 | Partial Confluence and Concurrency Control | 232 |
| 7.2.3 | Concurrency by transformation | 232 |
| 7.2.4 | Concurrent Objects | 233 |
| Bibliography | | 235 |
| Index | | 245 |

Acknowledgments

First of all, I would like to thank my supervisor David Walker without whom this work would not have been possible. I am indebted to him for acquainting me with concurrency theory as well as for teaching me the practice and purpose of research. Throughout the development of this thesis he was always willing to discuss my work and with his original ideas he guided me through the arguments of the thesis.

Furthermore, I am very grateful to the Department of Computer Science, University of Warwick, for employing me as a Teaching Assistant during my Ph.D. studies, providing me with financial support and a valuable experience. Many thanks also go to my colleagues in the department for contributing towards a stimulating and friendly working environment, and Xinxin Liu for interesting discussions while he was at Warwick. Drafts of chapters of the thesis were read by Leslie Goldberg and Michael Luck and I am grateful for their useful suggestions on presentation.

I would like to thank my friends for their encouragement and for making my time at Warwick enjoyable - in particular, Niki, Tommaso, Irena, Mike and Paul, and finally, my parents for the support they have given me over the years.

★ ★ ★

This is a revised version of my thesis; I have benefited greatly from the constructive criticism of my examiners Mathai Joseph and Frits Vaandrager.

Declarations

**This thesis was composed by myself and the work is my own unless otherwise stated.
Material from the thesis has been published in conference proceedings.**

Abstract

This thesis is concerned with the specification and verification of mobile systems, i.e. systems with dynamically-evolving communication topologies. The expressiveness and applicability of the π_v -calculus, an extension of the π -calculus with first-order data, is investigated for describing and reasoning about mobile systems.

The theory of confluence and determinacy in the π_v -calculus is studied, with emphasis on results and techniques which facilitate process verification. The utility of the calculus for giving descriptions which are precise, natural and amenable to rigorous analysis is illustrated in three applications. First, the behaviour of a distributed protocol is analysed. The use of a mobile calculus makes it possible to capture important intuitions concerning the behaviour of the algorithm; the theory of confluence plays a central rôle in its correctness proof. Secondly, an analysis of concurrent operations on a dynamic search structure, the *B*-tree, is carried out. This exploits results obtained concerning a notion of partial confluence by whose use classes of systems in which interaction between components is of a certain disciplined kind may be analysed. Finally, the π_v -calculus is used to give a semantic definition for a concurrent-object programming language and it is shown how this definition can be used as a basis for reasoning about systems prescribed by programs. Syntactic conditions on programs are isolated and shown to guarantee determinacy. Transformation rules which increase the scope for concurrent activity within programs without changing their observable behaviour are given and their soundness proved.

Glossary

The following table summarizes the main notations used in the thesis. The page numbers refer to the page where the notation is first used.

PROCESS CONSTRUCTIONS

| | | |
|--|-----------------------------------|----|
| a, b | names | 14 |
| t | data term | 15 |
| $t * \ell_j$ | field selector | 15 |
| α | prefix | 16 |
| $t(\tilde{u}), \bar{t}(\tilde{t}), \tau$ | input, output and silent prefixes | 16 |
| D | agent constant | 16 |
| $D \stackrel{\text{def}}{=} (\nu \tilde{u})P$ | constant definition | 17 |
| $D(\tilde{u})$ | constant application | 16 |
| F | process abstraction | 16 |
| Pr | set of process expressions | 16 |
| P, Q | process agents | 16 |
| $\sum_{i \in I} \alpha_i. P_i$ | process sum | 16 |
| $P \mid Q$ | parallel composition | 16 |
| 0 | inactive agent | 16 |
| $(\nu x)P$ | restriction | 16 |
| $\text{cond}(t_1 \triangleright P_1, \dots, t_n \triangleright P_n)$ | conditional agent | 16 |
| $!P$ | replication | 18 |
| $\text{fn}(P), \text{bn}(P)$ | free and bound names of P | 17 |
| $\text{obj}(\alpha), \text{subj}(\alpha)$ | object and subject of α | 16 |

PROCESS SEMANTICS

| | | |
|------------------------------|---|-----|
| $=$ | α -equivalence | 17 |
| \equiv | structural congruence | 19 |
| Act | set of observable actions | 22 |
| Act^+ | set of actions | 22 |
| μ | action | 22 |
| $\xrightarrow{\mu}$ | labelled transition | 22 |
| $fn(\mu), bn(\mu), n(\mu)$ | free names, bound names and names of μ | 23 |
| $\alpha \text{ comp } \beta$ | complementary actions | 23 |
| $\dot{\sim}, \approx$ | strong and weak bisimilarity | 26 |
| $\dot{\sim}_b$ | branching bisimilarity | 28 |
| $=_{cs}$ | coupled equivalence | 30 |
| $=_{cw}$ | weak coupled equivalence | 30 |
| $\dot{\sim}_\downarrow$ | d-bisimilarity | 32 |
| $\dot{\sim}_\downarrow^b$ | db-bisimilarity | 32 |
| $\dot{\sim}_{tr}$ | trace-equivalence | 43 |
| $\dot{\sim}_\downarrow^R$ | db^R -bisimilarity | 106 |
| $\alpha \beta$ | the weight of α over β | 49 |
| s/α | the excess of s over α | 54 |
| L | sort | 14 |
| L^+ | set of actions with positive subject in L | 23 |
| L^- | set of actions with negative subject in L | 23 |
| L^\pm | $L^+ \cup L^-$ | 23 |
| $cc(P)$ | the convergent core of P | 63 |
| $\mathcal{T}[IV]$ | the pruning operation | 75 |
| P^R | pruned transition system of P | 75 |
| $\mathcal{T}^{\leq 1}$ | pruned state space of \mathcal{T} | 103 |
| P^b | pruned transition system of P | 103 |

Chapter 1

Introduction

Concurrency is one of the most challenging areas of research in computer science. The construction, description and analysis of concurrent systems is a difficult task. The overall behaviour of a concurrent system is the result of the interactions of its individual components and their consequent evolution. Many threads of control may be present and several interactions may take place at the same time, and this can give rise to subtle and often unanticipated, indeterminate behaviour.

The development of mathematical frameworks for reasoning about concurrent systems has been a major goal of research. Many models and methods have been proposed in the past few decades. One of the most successful among them has been *process calculus*. Robin Milner's CCS [Mil80, Mil89] is generally accepted to be the initiator of this approach. It, and similar theories such as ACP [BW90], CSP [Hoa85], and Meije [Bou85], model the pure communication and synchronization aspects of concurrent systems with the aid of a small selection of basic constructors each representing a distinct idea. A limitation of such calculi is that the set of channels via which a term of the calculus may communicate with its environment is completely determined by its syntax and remains fixed during computation. Effectively, this prevents the notion of *mobility* from being captured in the calculus.

There have been a number of attempts to introduce mobility into process calculus. The most prominent of these is the π -calculus of Milner, Parrow and Walker [MPW92]. This is a calculus of communicating systems in which one can naturally express processes which have changing structure. Not only may the component agents of a system be arbitrarily linked, but a communication between neighbours may carry information which changes that linkage. Central to the π -calculus is the notion of *name*. Processes which share the acquaintance of a name may use it to communicate with each other and, as names may be passed as objects of in-

teractions, name acquaintances, and thereby the capability to communicate with other processes, can be acquired. This gives the calculus a much greater expressiveness than CCS. For instance, there have been encodings into the π -calculus of the λ -calculus [Mil92a], agent-passing process calculi [Tho90, San92, Ama93], and concurrent-object languages [Wal91, Jon93b]. Furthermore, it has been used to model communication protocols [OP92, Ora94] and data structures [Mil92b]. Over the last six years it has been a subject of intense research and its mathematical theory has been substantially developed while automated tools [Vic94] have been constructed to aid with reasoning about π -calculus processes.

The purpose of this thesis is to experiment with a variant of the π -calculus for describing and reasoning about mobile systems. The calculus under study, the π_v -calculus, is an extension of the polyadic π -calculus which accommodates data values other than names and a kind of conditional agents. We illustrate how it can be used for giving precise and natural definitions for a variety of systems. Then we employ the process-calculus descriptions as a basis for reasoning rigorously about the systems in question. A major benefit of such a study is that it allows us to observe the mobile process calculus and its associated techniques in action and thus to draw conclusions regarding their usefulness, limitations and applicability. Furthermore, it allows the distillation of common patterns of behaviour of mobile systems worthy of systematic analysis and leads to the refinement of process-algebraic techniques. In addition, the thesis contains a study of the notion of determinacy and confluence in the mobile setting, with emphasis on the development of results and techniques regarding these and related notions which can be used to facilitate process verification. Indeed, the theory developed plays a central rôle in the analysis of all three of the applications we consider.

In the next section, we give a brief overview of the major attempts to encapsulate mobility in models of concurrency. We then proceed to discuss the main motivations and directions followed in the thesis. In the subsequent presentation, it is assumed that the reader is familiar with process calculi such as CCS and we refer to [Mil89] for background information.

1.1 Mobility

There have been a number of concurrency formalisms which allow mobility other than the π -calculus. However, their theory has not yet been fully developed. The first of these is Hewitt's *actor systems* [Cli81, Agh86]. An actor is an active object which has acquaintances to other actors and can communicate with them via

asynchronous messages which are themselves actors. New actors may be created as computation proceeds and the acquaintances of an actor may change. Thus systems with dynamically-evolving structure can be expressed. The actor model has been fairly successful in reasoning about mobile systems and it has been used in studying computer architectures and object-oriented languages. However, mathematical treatment of its basic concepts has not been elaborated and notions of behavioural equivalences have not been explored, especially when compared with analogous work on process calculi.

An algebraic formulation of mobility was first achieved by Engberg and Nielsen [EN86]. Their Extended CCS (ECCS) featured mobility by using label expressions to capture parametric channels. However, the somewhat difficult treatment of constants, variables and channels hindered its success. The π -calculus was strongly influenced by that work. Features such as the semantic treatment of scope extrusion and the extension of the definition of bisimulation, originated from the work on ECCS. However, by removing the distinction between ECCS entities, the π -calculus yields an elegant and uniform framework where only names and agents are involved. A refinement of the π -calculus, the polyadic π -calculus, was presented in [Mil92b] where communication of tuples was added and a notion of typing of channels was introduced. Other work on models for processes without the restriction of a fixed initial connectivity includes the DyNe formalism of Kennaway and Sleep [KS85], which was developed specifically to describe parallel graph reduction in the context of a project to design a parallel processor, and work on parametric channels by Astesiano and Zucca [AZ84].

These models achieve mobility by enriching the handling of a channel. By contrast, an alternative approach for achieving mobility, often referred to as the higher-order approach, is by transmitting processes as messages. It was studied by various authors, including Astesiano and Reggio [AR87] in the setting of algebraic specification, Boudol [Bou89] in the context of the λ -calculus, and Nielson [Nie89] with emphasis on types. In addition, Thomsen's Calculus of Higher Order Communication (CHOCS, [Tho90]) is an extension of CCS where the content of communication are processes, while Sangiorgi's Higher-Order π -calculus ($HO\pi$, [San92]) enriched the π -calculus to allow the communication of process abstractions. The latter author also compared the approaches and showed that such an extension does not add expressiveness to the π -calculus. Thus the first-order paradigm, which enjoys a simpler and more approachable theory, can be taken as a basic.

1.2 Determinacy and Confluence

Many concurrent systems are inherently indeterminate. Consequently, general models of concurrent systems, and in particular semantic accounts of concurrent programming languages, are more complex than those for sequential systems and languages. However, in constructing a concurrent system or program it is often the intention that it will operate in a ‘well-behaved’ way. In particular, it might be expected that the system will be predictable, in the sense that if the same experiment is run twice from the initial state of the system then, each time, the system will exhibit the same behaviour. Such ideas motivated the study of the notions of determinacy and confluence in concurrent systems, initiated in process calculi by Milner in [Mil80, Mil89] and subsequently elaborated by other authors [San82, Tof90, GS95, Nes96]. (For background information on determinacy and confluence see Chapter 11 of [Mil89] and for the precise definitions of the notions see Chapter 3 of this thesis). The theory has been employed in various contexts. For example, in [GS95, Mil89] it was employed for protocol verification, while in [HP94, Qin91] it was used for on-the-fly reduction of finite state spaces. Furthermore, in [Tof90, Nes96], it was studied and discussed within the context of concurrent programming.

There appear to be two main benefits of a rigorous study of determinacy and confluence for concurrent systems. First, given that determinate behaviour may be significant for a variety of systems, it appears worthwhile to consider the development of a compositional method for the construction of determinate and confluent systems. Indeed, a main motivation behind the theory developed in [Mil89] was to provide a theoretical framework within which one may build determinate systems from determinate components. Secondly, determinacy and confluence of systems may be systematically exploited in formal verification. For instance, one of the main observations made in [Mil80] was that trace equivalence coincides with bisimilarity for confluent CCS agents. Furthermore, confluent agents are semantically invariant with respect to internal actions. These facts considerably ease the task of verifying confluent systems.

In this thesis we perform a rigorous analysis of determinacy in the context of the π_v -calculus. We explore and compare a number of different formalizations of the notion, each built on a distinct behavioural equivalence, and we investigate their theory. We also study in some depth which of the calculus constructors preserve confluence and distinguish constructs that may be a source of indeterminacy. We observe that indeterminate behaviour may be caused by competition of compo-

nents of a system to access the same resource. For example, suppose that at some point during the computation of a certain system, two of its components are able to communicate via a channel with a third. The subsequent behaviour of the system will, in general, not be independent on which interaction takes place and hence, the behaviour of the system is unpredictable. This possibility is not specific to mobile systems. Rather, it is a source of indeterminacy for concurrent systems in general. However, to prevent such situations from arising requires special consideration in a mobile setting since the topology of a system does not remain fixed during computation. Thus, given a mobile agent, it is not sufficient to impose conditions on the capabilities of using names solely on its initial state. It is also necessary to ensure that names are used in a disciplined way so that sharing may never arise. Other features specific to the calculus under study are scoping of names and name instantiation. We address these in detail. A key observation is that in reasoning about the behaviour of a confluent system it is often sufficient to examine in detail only a part of its behaviour: from this and the system's confluence it is then possible to deduce properties of the remaining behaviour. This observation is exploited in many instances in the thesis.

It is often the case that systems have some good behavioural properties but fall short of being confluent. Various authors have been concerned with the investigation of such properties which can be captured formally and exploited for process verification. With the aim of identifying conditions under which combinations of non-confluent agents yield confluent systems, a study of notions of semi-confluence was undertaken in [Tof90]. Another notion of partial confluence was introduced in [LW95a]. Like semi-confluence, the essence of this notion is that the occurrence of some actions will not preclude the occurrence of some others. However, a fundamental difference of partial confluence from the earlier notions of (semi) confluence is that it does not require agents to be semantically invariant under silent actions. The motivation behind its definition was to study systems composed of non-confluent but well-behaved components, capable of state-changing internal actions. Its theory was developed and it was applied to prove the indistinguishability within an arbitrary program context of two classes expressed in a concurrent-object programming language.

In the thesis we consider a number of extensions to the theory of partial confluence in the context of the π -calculus. The first is concerned with extending the partial confluence theory of [LW95a] to take explicit account of divergence. The motivation behind the development was provided by our study of a concurrent-object language where the theory is applied to prove the correctness of transformation rules

for concurrent-object programs. In this context, divergence is considered to be an undesirable property of the systems studied and thus it is appropriate to employ a notion of observation equivalence that takes account of infinite silent paths.

A key insight provided by the partial confluence theory of [LW95a] is that when reasoning about partial-confluent systems in which interaction is of a certain disciplined kind, it is sufficient to examine only a part of their behaviour. The systems considered can be viewed as consisting of a client and a server which interact in a question-answer fashion, with possibly many questions outstanding at any moment. An important characteristic of these systems is that on accepting a question from the client, the server immediately assumes a state in which the answer to that question is determined. Moreover, it is assumed that when a question is invoked in the client, the name which is supplied for return of an answer is distinct from answer names supplied in other question invocations. This form of question-answer interaction is quite common in computer systems. However, it is often the case that the processing of a question by a server results in a change of the server's state. For example, the commitment to perform a write of a register in a multiple-read single-write memory alters the state. We consider two significant generalizations of the theory of [LW95a] which relax these two assumptions and prove that analogous results hold.

1.3 Concurrency Control of *B*-trees

Concurrent access to complex data structures and, in particular, structures which serve as indices to database systems has long been an area of active research in the database community. One of the main challenges has been to design structures and their associated operations which provide a high degree of concurrency while preventing interference among concurrent processes and efficiently support multiple users. Furthermore, such structures should be able to handle growth and to rebalance themselves as the amount of data stored increases and decreases. A variety of such dynamic structures have been proposed in the literature, including dynamic hash data structures [Ell87, Lar78, Lom83] and binary search trees [KL80, ML84, Ell80], and a variety of algorithms have been proposed and implemented for them.

The structure we investigate in the thesis is the *B*-tree [BM72] which, along with its variants, is another widely used index structure. Concurrent operations on such structures have been studied in many papers, including [BS77, KW82, LY81, MS78, Sag86, Sam76, SG88]. We perform a rigorous study of concurrent operations

on a variant of the B^* -tree [Wed74]. This variant is the B^{link} -tree of [LY81], and the operations are those of [Sag86] which improve on the operations given in [LY81]. In [LY81] and [Sag86], the structure is described and explained using prose and diagrams, the algorithms are described using pseudo-code, and the arguments for their correctness are informal though detailed.

It is generally accepted that it is often very difficult to construct concurrent systems and give convincing arguments that they satisfy desired properties. Indeed, without a formal model of the system in question, it can be difficult to even express precisely what the properties are. In the case of database concurrency control many varieties of 'serializability' have been proposed and used as criteria by which to judge algorithms, each suitable for different settings. For instance in [Sag86], the notion of 'data serializability' was adopted which requires that any 'schedule of the operations' arising by executing the algorithms is 'data equivalent to a serial schedule' and 'preserves the validity of the data structure'.

By using the π_v -calculus as a basis of our analysis, we may give direct and succinct descriptions of the operations and the underlying data structure which capture naturally its dynamically evolving interconnection. In addition, we are able to assert the criterion of correctness of the system in terms of its observable behaviour, in contrast with forms of serializability. This is achieved as follows: we define a simple agent which captures the intended behaviour of the system and assert that the agent-description of the system and the simple agent are behaviourally equivalent. The principle that concurrent systems should be compared on the basis of their observable behaviours is widely held and has been studied in depth in concurrency theory, beginning with [Mil80]. It has also been explored extensively and has been argued to be sound specifically in the area of database concurrency control in a study of atomic transactions in [LMWF94]. That study employs I/O-automata, which are themselves very closely related to process calculus [Vaa91], to give a uniform analysis of a great variety of concurrency control algorithms.

In our view, the use of a mobile process-calculus for reasoning about such algorithms has two significant benefits. First, unlike I/O-automata, it allows a direct representation of changing structure in systems and, secondly, by acting as unifying framework for the description of a variety of systems, it allows concepts and techniques developed for the purpose of one application to be generalized and applied in others. Indeed, verifying the correctness of the B -tree algorithms is significantly facilitated by the use of the theory of partial confluence. In fact, the theory enables us to prove that the full system, in which several search processes may be accessing a node of the tree concurrently, has the same behaviour as two simplifications of

it: when each node allows at most one access at a time, and when the whole tree allows at most one operation to be active at any time. These observations allow us to obtain results about the full system by analysing the simpler systems.

1.4 Object-oriented languages

The notions of *name* and *mobility* are common in many areas of computer science. One such area, much studied in the last few years, is that of *concurrent object-oriented programming*. A program described by concurrent object-oriented languages prescribes a system containing a collection of objects which refer to each other by using names. As computation proceeds new objects may be created and the linkage among objects may change with the passing of names among objects. Thus, the ability of the π -calculus to directly encode such features makes it promising for giving semantics to these languages.

The idea of using a process calculus to give a semantic definition to a concurrent programming language was first studied in [Mil80, Mil89], where a simple concurrent programming language was given a semantics by translation to CCS. This method was further explored in [Pap92] where the use of CCS for giving semantics to sequential object-oriented languages was studied while in [Vaa90] the parallel object-oriented language POOL was given a translation to the language of ACP. Furthermore, semantics were given by translation to the π -calculus and its variants: for instance, π -calculus [Wal91, Jon93b, Wal95a], Higher-Order π -calculus [Wal95b], CHOCS [Tho93], π -calculus with simple values [Wal94, PW95], and an extension of π -calculus with higher-order abstractions and data other than names but with only first-order interaction [LW96]. The idea of using mobile calculi as a framework for studying object-oriented languages was a motivation behind the development of the Pict programming language [PT94]. This is a language based on the π -calculus within which one may express concurrent objects.

In the thesis we build on work using mobile process calculi as semantic bases for concurrent-object languages. In our view these general models of concurrent systems with changing structure are very well suited to giving natural and direct semantic definitions of such languages. Our main motivation is to experiment with the semantic definition for a specific concurrent-object language in order to reason rigorously about programs expressed in it. Indeed, a major reason for experimenting with mobile process calculi for giving semantics to object-oriented languages is the desire to employ their theory to prove behavioural properties of systems prescribed by programs and to improve the understanding of language features. At the same

time, the intention is to investigate the calculi themselves, to develop their theory and explore how they can be refined and enriched in order to capture more successfully the features of the systems under study and thus facilitate their specification and verification.

We begin our analysis by studying the notion of determinacy within the language and we isolate syntactic conditions which guarantee that programs conforming to them are determinate. As demonstrated in our study of determinacy and confluence in the π_v -calculus, in order to achieve this, it is sufficient to ensure that in no state reachable from the initial configuration of a program do two objects share the ability to communicate with a third. Thus we seek syntactic conditions on programs which ensure that references to objects may not be shared. To achieve this without impoverishing the language too severely we augment it with an expression form which expresses a destructive read of a variable. This gives a means of expressing cleanly that when one object sends to a second object a reference to a third object, the first object relinquishes the reference to the third. The proof of the claim that the conditions guarantee determinacy of programs is carried out on the basis of the translational semantics to the π_v -calculus. It turns out that the translation of a program satisfying the conditions is an agent which falls into one of the classes that our process-calculus study has categorized as determinate and even confluent.

Motivation for the second part of our study was provided by the work of C B Jones on the formal development of concurrent programs utilizing ideas from object-oriented programs. A central part of the development process was the application of transformations to increase the scope of concurrent activity within programs without altering their observable behaviour [Jon93a, Jon93b]. In these papers, Jones suggested the π -calculus as a basis for proving the correctness of these transformations because of its rich algebraic theory and the directness of the semantics it provides to the language under study. He proceeded to give some arguments for the correctness of an application of the transformations and raised the challenging question: under which conditions may the transformation rules be safely applied, and how may the existing techniques for reasoning about programs be extended and used to prove their soundness? A central part of this challenge involved formalizing an appropriate correctness criterion for the soundness of the transformations. Briefly, this should identify program fragments which, although exhibiting different observable behaviours, cannot be distinguished by program contexts. A solution to this problem was provided by Liu and Walker in [LW95a], where the correctness of an instance of the transformations was established. This involved proving the

indistinguishability of two classes of a concurrent-object language within arbitrary program contexts and it was achieved by capturing important properties satisfied by the classes and legal program contexts. Here our aim is to enunciate syntactic conditions under which the transformations may be applied and prove that this is the case. Generalizations of the syntactic conditions which guarantee determinacy play a rôle in the rules we consider. Furthermore, the theory of confluence and partial confluence is central to the proof of correctness. In particular, we observe that in order to reason about the behaviour of the systems in question it is sufficient to examine in detail only a part of their behaviour; from this and the fact of the system's (partial) confluence, it is possible to deduce properties of the remaining behaviour. We employ the theory of partial confluence accommodating divergence. This turns out to be necessary as, in considering the correctness of the transformation rules, the possibility of non-termination of method invocations must be taken into account.

Various other theoretical frameworks have been developed and used for studying object-oriented languages. Within the context of the process-calculus approach, various authors have used extensions of (mobile) process calculi by adding objects and functional constructs. Work in this area includes Oscar Nierstrasz's *Object Calculus* [Nie91]; furthermore, [Vas94, KY94, San96] concentrate on interpreting typed objects in process calculi. A different possibility for studying concurrent objects was offered by the actor model. This has been used for the design of concurrent object-oriented languages (e.g. ABCL [Yon90]).

Recently, an application-oriented approach was undertaken in the development of functional object calculi [AC94b, AC94a, AC96]. These calculi, which have primitives for object-oriented features, have been used to account for a range of object-oriented concepts and have led to the design of object languages (Obliq [Car95]). All such calculi developed until now are sequential. However, the approach appears to be very promising and it will be interesting to see it extended to the concurrent setting.

1.5 Summary of the thesis

We begin Chapter 2 with an introduction to the π_v -calculus. We present its syntax and its operational semantics and we proceed to impose a sorting discipline on its terms. We continue with a review of aspects of observational equivalence and divergence which we will employ in the thesis. Finally, we introduce some useful properties of agents concerning the use of names, and we study sufficient conditions

which guarantee that the properties are enjoyed.

In Chapter 3, we carry out a study of the notions of determinacy and confluence in the π_v -calculus. One of the aims is to derive as simple characterizations for them as possible, as this will facilitate the confirmation of their satisfaction by systems. We present three notions based on weak bisimilarity, branching bisimilarity and a divergence-sensitive variant of weak bisimilarity. The three notions are closely related and, in fact, the ones based on the first two bisimilarities coincide. We investigate their preservation by combinators of the calculus and observe that confluence, which is a stronger notion than determinacy, is preserved by a reasonably large class of restricted compositions. We illustrate the theory of confluence and the π_v -calculus with a substantial example: a verification of a protocol taken from [Vaa95]. The proof uses a confluence argument which considerably reduces the complexity of the problem and is facilitated by the use of sorts. Moreover, it employs mobility to dynamically represent intuitive ideas concerning the protocol.

In Chapter 4, we deepen the analysis of confluence to the partial case. First, we study a notion of partial confluence which is sensitive to divergence. We develop some of its theory and extend the results of [LW95a] to this setting. Then, we present two extensions of these results to encompass a wider range of systems. We refer to the set of properties required of the agents under study as *social confluence*. It turns out that in order to achieve one of the extensions it is necessary to refine the notion of partial confluence to a notion which although not implying determinacy, can nonetheless be argued to be well-behaved.

In Chapter 5, we perform a rigorous analysis of concurrent operations on B -trees. We model the system consisting of the data structure and the operations as an agent. The criterion of correctness is formalized in terms of the observable behaviour of the system under study. More specifically, we define a simple agent whose observable behaviour describes the expected interactions of the system and the environment, and assert that the two systems are behaviourally equivalent. We consider two possible interpretations of ‘behavioural equivalence’ and present their respective correctness results. These make use of the two versions of the theory of social confluence. The weaker notion assumes that operation instances can be distinguished from each other: when a method is invoked the name which is supplied for the return of the answer is distinct from answer names supplied in other operation invocations. This notion is adequate for the type of the application and its correctness proof can be obtained by making use of the simpler version of the theory of social confluence. Nonetheless, we can show that a full behavioural equivalence holds by employing the extension of the theory. We continue by proposing an

improvement to one of the operation algorithms and outline how the existing analysis can be extended to verify its correctness.

In Chapter 6, we present a semantics by translation to the π_v -calculus of a concurrent-object language and employ this to reason rigorously about programs written in the language. First, we present syntactic conditions which guarantee that programs conforming to them are determinate and prove that this is the case by analysing the agents representing the programs in question and verifying their confluence by appealing to results of Chapter 3. Subsequently, a generalized version of these conditions plays a rôle in the transformation rules we consider. We prove that the transformations increase the scope for concurrent activity within systems prescribed by programs without altering their observable behaviour.

Finally, in Chapter 7, we comment on the results obtained and we present directions for possible future work.

Chapter 2

Background

In this chapter we review the basic concepts of the mobile process calculus we study in the thesis, the π_v -calculus. This is an extension of the polyadic π -calculus of [Mil92b] with data values other than names. We begin with a presentation of its syntax and then proceed to its operational semantics given in terms of an early transition system. The calculus is similar to that of [LW96], but restricted to a simpler setting: the calculus we consider does not have higher-order process abstractions and thus is a subset of the calculus introduced in [LW96]. For this chapter, it is desirable that the reader has some familiarity with (mobile) process calculi such as CCS [Mil80], since the presentation of the calculus hereby given is intense although self-contained.

We impose a sorting discipline on the calculus. Unlike the traditional disciplines associated with polyadic mobile calculi in the literature, we consider a *polymorphic system*. Although the system does not guarantee absence of run-time mismatches in general, we show that it is ‘safe’ for a certain subset of the language of the calculus. An important reason for its adoption is that it assists significantly the application of the calculus for the specification and analysis of systems.

We continue with a review of aspects of bisimulation and divergence and conclude the chapter with some useful properties of process communities. First, let us introduce some general notation for the thesis.

NOTATION:

- We let \mathcal{B}, \mathcal{R} range over relations. We write $(P, Q) \in \mathcal{B}$ and $P\mathcal{B}Q$ interchangeably.
- \mathcal{B}^{-1} and $\mathcal{B}\mathcal{B}'$ denote the inverse of \mathcal{B} and the composition of \mathcal{B} and \mathcal{B}' respectively.

- \emptyset is the empty set and \cup, \cap are union and intersection on sets. Further, for sets S_1 and S_2 , we write $S_1 - S_2$ for $\{x \in S_1 \mid x \notin S_2\}$. We often abbreviate $S \cup \{x\}$ to $S \cup x$, and similarly for \cap and $-$.
- We write \tilde{x} for a tuple $x_1 \dots x_n$ of syntactic entities and use ε for the empty sequence. Given $\tilde{x} = x_1 \dots x_n$, with $n \geq 1$, we write $\text{hd } \tilde{x}$ for x_1 , $\text{last } \tilde{x}$ for x_n , $\text{tl } \tilde{x}$ for $x_2 \dots x_n$ and $|\tilde{x}|$ for the length of \tilde{x} , n . Further, we write $|\tilde{x}|_a$ for the number of occurrences of a in \tilde{x} .
- For a set K , K^* denotes the set of all finite sequences of elements of K . Given a sequence s we write $s \upharpoonright L$ for the sequence obtained by projecting s on L . We say that t is a prefix of s , $t \leq s$, if $s = ts'$ for some sequence s' , where juxtaposition is concatenation. Given $s \in K^*$ and $a \in K$, we write $s - a$ for the sequence obtained by removing the first occurrence of a from s if it exists and s otherwise.

2.1 The π_v -calculus

In this section we present the π_v -calculus, an extension of the polyadic π -calculus which accommodates data values other than names and a kind of conditional agents. The main concern in the description is to keep the calculus as simple as possible at the expense of making some strong assumptions.

We begin the presentation of the calculus by describing its data. We assume a set of *base types*, ranged over by B , among which is the Boolean type *bool*. We interpret each base type as a domain which we also denote by B . For each value c of a base type B we assume the existence of a *constant symbol* which denotes c . So for example, *true* and *false*, *nil* are constant symbols of the calculus, the last representing the undefined value for the types B . Moreover, we assume a set of *names*, \mathcal{N} , and a set of *labels*, \mathcal{L} , and we let x, y, a, b range over names and ℓ over labels. We assume a set of *name sorts*, ranged over by L and a single *label sort*, Λ . Each name has a name sort associated with it and each label is associated with Λ . If a name x is associated with a sort L , we say that x is of sort L and write $x : L$. We assume infinitely many names of each sort. Formally, the *first-order types* of the calculus, ranged over by i , are given as follows:

$$i ::= B \mid L \mid \{\ell_1 \vdash i_1, \dots, \ell_n \vdash i_n\}$$

where $\ell_1 \dots \ell_n$ are pairwise distinct labels. We refer to the type $\{\ell_1 \vdash i_1, \dots, \ell_n \vdash i_n\}$ as a *record type* with *labels* $\ell_1 \dots \ell_n$ and *component types* $i_1 \dots i_n$.

We assume a set of variables \mathcal{V} , ranged over by z , each of which has associated with it one of the base types, or one of the record types. Further, we assume the existence of *function symbols*, ranged over by f . Each such symbol has an arity $B_1 \times \dots \times B_n \rightarrow B$ where B_i and B are base types and is interpreted as a function $f : B_1 \times \dots \times B_n \rightarrow B$. Combinations of these constructions give rise to the data terms of the calculus. Ranged over by t , the terms and their associated types (we write $t : i$ if t has type i) are given inductively as follows:

- the names, $x \in \mathcal{N}$, where if x is of sort L then $x : L$;
- the variables, $z \in \mathcal{V}$, where if z is of type i then $z : i$;
- constant symbols, where if c is of type B then $c : B$;
- function applications, $f(t_1, \dots, t_n)$, where if $f : B_1 \times \dots \times B_n \rightarrow B$ and $t_i : B_i$, then $f(t_1, \dots, t_n) : B$;
- record terms, $\{\ell_1 := t_1, \dots, \ell_n := t_n\}$, where if $t_j : i_j$ then $\{\ell_1 := t_1, \dots, \ell_n := t_n\} : \{\ell_1 \vdash i_1, \dots, \ell_n \vdash i_n\}$;
- field selector terms, $t * \ell_j$, where if $t : \{\ell_1 \vdash i_1, \dots, \ell_n \vdash i_n\}$ then $t * \ell_j : i_j$.

The *values*, ranged over by v are the following data terms:

$$v ::= x \mid c \mid \{\ell_1 := v_1, \dots, \ell_n := v_n\}$$

A data term is *closed* if it contains no variables. The *evaluation relation* \rightarrow from closed data terms to values is given inductively as follows:

- $x \rightarrow x$ and $c \rightarrow c$;
- if $t_1 \rightarrow c_1, \dots, t_n \rightarrow c_n$ and $f(c_1, \dots, c_n) = c$, then $f(t_1, \dots, t_n) \rightarrow c$;
- if $t_1 \rightarrow v_1, \dots, t_n \rightarrow v_n$ then $\{\ell_1 := t_1, \dots, \ell_n := t_n\} \rightarrow \{\ell_1 := v_1, \dots, \ell_n := v_n\}$;
- if $t \rightarrow \{\ell_1 := v_1, \dots, \ell_n := v_n\}$ then $t * \ell_j \rightarrow v_j$.

It is straightforward to prove that \rightarrow is a function from closed data terms to values and that if $t : i$, $t \rightarrow v$ then $v : i$. The following notation is useful:

Notation 2.1.1 Given x_1, \dots, x_n of types T_1, \dots, T_n we write $\text{sort}(\tilde{x})$ for \tilde{T} .

The *agent types* of the calculus, ranged over by ξ , are given by

$$\xi ::= \text{abs}(i_1, \dots, i_n).$$

We assume a set of *agent constants* ranged over by D . Each such constant has an associated agent type, and we assume the existence of an infinite number of constants of each type. The set Pr of *process expressions*, ranged over by P and Q , and the *abstractions*, ranged over by F , are given as follows:

$$\begin{aligned} P &::= \sum_{i \in I} \alpha_i. P_i \mid P \mid Q \mid (\nu x)P \mid F(\tilde{t}) \mid \text{cond}(t_1 \triangleright P_1, \dots, t_n \triangleright P_n) \\ F &::= (\tilde{u})P \mid D \\ \alpha &::= \tau \mid t(\tilde{u}) \mid \bar{t}(\tilde{t}) \end{aligned}$$

where x ranges over names and u ranges over names and variables. We refer to process expressions and abstractions as *agents*. The prefixes $t(\tilde{u})$ and $\bar{t}(\tilde{t})$ are called *input* and *output* prefixes respectively, while τ is referred to as the *internal prefix*. We say that a prefix α occurs *guarded* in a process Q if each occurrence of α in Q is within some subexpression $\beta.P$ where $\alpha \neq \beta$. Otherwise α occurs *unguarded* in Q . Similarly, we say that a name x occurs guarded (resp. unguarded) in Q if it appears within a prefix t which occurs guarded (resp. unguarded) in Q . Moreover, given input and output prefixes $t(\tilde{v})$, $\bar{t}(\tilde{t})$ we refer to t as the subject of the prefixes and to \tilde{v} , \tilde{t} as the object of the input and output prefixes respectively. Further, we say that in the input prefix, t occurs in *positive* subject position (or is a *positive subject*), whereas in the output prefix it occurs in *negative* subject position (or is a *negative subject*). We make use of the following notation.

Notation 2.1.2 We write $\text{subj}(\alpha)$ for the subject and $\text{obj}(\alpha)$ for the object of a prefix α . For convenience we also write $\text{subj}(\tau) = \tau$.

We make the following assumptions and abbreviations on the above expressions. We assume that the elements of \tilde{u} are pairwise distinct. In summations, the set I is finite and we use the symbol 0 to refer to the empty summation; we also write $P_1 + P_2$ for the binary summation $\sum_{i \in \{1,2\}} P_i$ and we often abbreviate $\alpha.0$ to α and $(\nu a)(\nu b)P$ to $(\nu ab)P$. Note that if a tuple is empty then the brackets $()$ or $()$ are omitted.

The formal semantics of the calculus is presented in a later section. In the meantime we make some general remarks and define some relevant notions. We begin by noting that the operators $t(\tilde{u}).P$, $(\tilde{u})P$ and $(\nu x)Q$ bind all free occurrences of the elements of \tilde{u} in P and of x in Q . These binders give rise to the sets $\text{fnv}(P)$

and $\text{bnv}(P)$ of *free and bound names and variables* of an agent P , defined in the following table.

| P | $\text{fnv}(P)$ | $\text{bnv}(P)$ |
|--|--|---|
| $\tau.Q$ | $\text{fnv}(Q)$ | $\text{bnv}(Q)$ |
| $t(\tilde{u}).Q$ | $\text{fnv}(t) \cup (\text{fnv}(Q) - \tilde{u})$ | $\text{bnv}(Q) \cup \tilde{u}$ |
| $\tilde{t}(\tilde{t}).Q$ | $\text{fnv}(t) \cup \text{fnv}(\tilde{t}) \cup \text{fnv}(Q)$ | $\text{bnv}(Q)$ |
| $\Sigma_{i \in I} P_i$ | $\bigcup_{i \in I} \text{fnv}(P_i)$ | $\bigcup_{i \in I} \text{bnv}(P_i)$ |
| $P_1 \mid P_2$ | $\text{fnv}(P_1) \cup \text{fnv}(P_2)$ | $\text{bnv}(P_1) \cup \text{bnv}(P_2)$ |
| $(\nu x)Q$ | $\text{fnv}(Q) - x$ | $\text{bnv}(Q) \cup x$ |
| $F(\tilde{t})$ | $\text{fnv}(F) \cup \text{fnv}(\tilde{t})$ | $\text{bnv}(F) - \text{fnv}(\tilde{t})$ |
| $\text{cond}(t_1 \triangleright P_1, \dots, t_n \triangleright P_n)$ | $\bigcup_{1 \leq i \leq n} (\text{fnv}(t_i) \cup \text{fnv}(P_i))$ | $\bigcup_{1 \leq i \leq n} \text{bnv}(P_i)$ |
| $(\tilde{u})Q$ | $\text{fnv}(Q) - \tilde{u}$ | $\text{bnv}(Q) \cup \tilde{u}$ |

For the free and bound names and variables on agent constants see below. The free names and variables of a term are given in the obvious way: $\text{fnv}(x) = x$, $\text{fnv}(z) = z$, $\text{fnv}(c) = \emptyset$ and so on. It is convenient to be able to distinguish between the free names and the free variables in a given sequence. We thus adopt the following notation:

$$\begin{aligned} n(\tilde{u}) &= \tilde{u} \cap \mathcal{N} \\ v(\tilde{u}) &= \tilde{u} \cap \mathcal{V} \end{aligned}$$

We extend this notion to agents and, given an agent P , we write

$$\begin{aligned} \text{fn}(P) &= n(\text{fnv}(P)), \text{ for the free names of } P, \\ \text{bn}(P) &= n(\text{bnv}(P)), \text{ for the bound names of } P, \text{ and} \\ n(P) &= \text{fn}(P) \cup \text{bn}(P), \text{ for the names of } P. \end{aligned}$$

We have the following definition.

Definition 2.1.3 An agent P *owns* a name x if $x \in \text{fn}(P)$.

Throughout the thesis we work up to α -conversion on names and variables: we identify process expressions and abstractions which differ only by change of bound names and variables, and we write $P = Q$ if P and Q are α -equivalent. Thus we may assume that all bound names/variables of a process are different from each other and from the free names/variables of the process.

We assume that each agent constant D has a *defining equation* $D \stackrel{\text{def}}{=} (\tilde{u})P$ where \tilde{u} contains all names and variables occurring free in P . Although this requirement is useful for giving simple inductive definitions of substitution and of the free

names of agents, we will sometimes omit mentioning all of the parameters where this is not necessary for the sake of clarity. For example, we may write $D \stackrel{\text{def}}{=} a(x). \bar{x}. D$, if we intend to maintain a for all uses of D .

Constants are employed to represent infinite behaviour: in the definition $D \stackrel{\text{def}}{=} (\tilde{u})P$, P may contain occurrences of constants including D itself. Thus D may be compared to a procedure where \tilde{u} are the formal parameters and $D\langle\tilde{t}\rangle$ is a ‘call’ of D . A necessary requirement in an application is that \tilde{t} has the correct type. This is ensured by the use of the sorting which we study in the following section. An agent constant we will refer to in subsequent sections is Ω ,

$$\Omega \stackrel{\text{def}}{=} \tau. \Omega.$$

Before we proceed, we present an additional form of process expression derivable from the syntax which we will often be using: $!P$ is defined by

$$!P \stackrel{\text{def}}{=} (\tilde{u})(P \mid !P),$$

where $\tilde{u} = \text{fnv}(P)$. This expression, usually referred to as a *replicator* intuitively represents an unbounded number of copies of P in parallel.

A *substitution* is a type-respecting partial function from names and variables to values. We write $\{\tilde{v}/\tilde{u}\}$ for the substitution that maps \tilde{u} to \tilde{v} (where \tilde{u} consists of pairwise distinct elements). Moreover, we write $\sigma(u)$ for the value of σ at u and $\sigma(\tilde{u})$ for $\{\sigma(u) \mid u \in \tilde{u}\}$.

Definition 2.1.4 Given a term t the *effect* of a substitution σ , written $t\sigma$, is defined inductively as follows:

$$\begin{aligned} x\sigma &= \sigma(x) \\ z\sigma &= \sigma(z) \\ c\sigma &= c \\ (f(t_1, \dots, t_n))\sigma &= f(t_1\sigma, \dots, t_n\sigma) \\ (\{\ell_1 := t_1, \dots, \ell_n := t_n\})\sigma &= \{\ell_1 := t_1\sigma, \dots, \ell_n := t_n\sigma\} \\ (t * \ell_j)\sigma &= t\sigma * \ell_j \end{aligned}$$

Further, the *effect* of substitution σ on an agent P , written $P\sigma$ is defined below. A substitution does not affect bound names. Thus to ensure that a free name in P does not become bound in $P\sigma$, we assume that all bound names of P have been α -converted to fresh names before the substitution is applied so that $\text{bn}(P) \cap \sigma(\text{fn}(P)) = \emptyset$.

$$\begin{aligned}
(F\langle\tilde{t}\rangle)\sigma &= F\langle\tilde{t}\sigma\rangle \\
((\tilde{y})P)\sigma &= (\tilde{y})(P\sigma) \\
(P_1 \mid P_2)\sigma &= (P_1\sigma) \mid (P_2\sigma) \\
((\nu x)P)\sigma &= (\nu x)(P\sigma) \\
(\tau.P)\sigma &= \tau.(P\sigma) \\
(t(\tilde{y}).P)\sigma &= (t\sigma)(\tilde{y}).(P\sigma) \\
(\bar{t}(\tilde{t}).P)\sigma &= \bar{t}\sigma(\tilde{t}\sigma).(P\sigma) \\
(\Sigma_{i \in I} P_i)\sigma &= \Sigma_{i \in I} (P_i\sigma) \\
(\text{cond}(t_1 \triangleright P_1 \dots t_n \triangleright P_n))\sigma &= \text{cond}((t_1\sigma) \triangleright (P_1\sigma), \dots, (t_n\sigma) \triangleright (P_n\sigma))
\end{aligned}$$

□

We adopt the following precedence among operators, in decreasing order: application, substitution, replication, prefixing, restriction, parallel composition, summation, abstraction.

Definition 2.1.5 *Structural congruence*, \equiv , is the smallest congruence over the class of π_v -calculus agents satisfying the following:

1. $P \equiv Q$ if $P = Q$.
2. $\Sigma_{i \in I} \alpha_i. P_i \equiv \Sigma_{j \in J} \alpha_j. P_j$, if J is a permutation of I .
3. the abelian monoid laws for \mid , $P \mid Q \equiv Q \mid P$, $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$, $P \mid 0 \equiv P$.
4. $(\nu x)(\nu y) P \equiv (\nu y)(\nu x) P$, $(\nu x) 0 \equiv 0$.
5. $(\nu x) P \mid Q \equiv (\nu x)(P \mid Q)$, if $x \notin \text{fn}(Q)$.
6. $(\nu x) P + Q \equiv (\nu x)(P + Q)$, if $x \notin \text{fn}(Q)$.
7. If $D \stackrel{\text{def}}{=} (\tilde{u})P$ then $D\langle\tilde{t}\rangle \equiv P\{\tilde{t}/\tilde{u}\}$.

□

2.2 Sorting

The notions of sort and sorting were first introduced in the π -calculus in [Mil92b]. A sorting system in the sense of [Mil92b] consists of a function from names to a set S of sorts and a function $ob : S \rightarrow S^*$, which stipulates that a name x of sort s may only

be used to send and to receive tuples of names whose sort is the tuple $ob(s)$. The main purpose of this device is to ensure that well-sorted (or well-typed) processes are free from run-time arity mismatches in interactions, such as $x(y).P \mid \bar{x}(z, w).Q$. This notion naturally extends to the typing of agents to ensure that the application of an abstraction is well-typed. For example, the application $D\langle 2 \rangle$ is well-typed for $D \stackrel{\text{def}}{=} (z)P, z : \text{int}$ whereas the application $D\langle 2, \text{true} \rangle$ is not.

Subsequently, various sortings have been proposed in the literature. These can be divided in two categories: the *by-name* approach (see [LW95b] for a discussion of the approach in a general setting), following Milner's approach described above, and the *by-structure* approach [San95, PS93, VH93]. In the *by-name* sorting, no two distinct types are considered to be equal. By contrast, in the *by-structure* setting a type is associated with a regular tree and two types are the same if their corresponding trees are the same. So, for example, if $x : s, y : s'$ and $ob(s) = ob(s') = (\text{int})$, then according to the *by-structure* approach s and s' are identified, whereas in the *by-name* approach they are not. In the sorting discipline we consider for the π_v -calculus, we follow the *by-name* approach. The main reason for this choice is that it allows us to impose distinctions among names which, although carrying similar data, have different rôles. Such knowledge may be useful in reasoning about processes, as will become apparent in later chapters.

The sorting system we employ is a special case of the system introduced in [LW95b] where the input and output sort of a name are identical. Thus, well-typed process expressions are defined via a function λ which associates each sort i with a set of tuples of types. Note that the system is polymorphic in nature (contrast λ with ob above). The intention is that if $\lambda(L) = \{\tilde{i}_1, \dots, \tilde{i}_n\}$ then a name of sort L may be used to communicate tuples of values of type \tilde{i}_j , for all j . For example, if $\lambda(L) = \{(\text{bool}), (\text{bool}, L)\}$ then a name of sort L may be used for communicating boolean values and pairs whose first component is a boolean and whose second component is a name of sort L .

Allowing a name to carry tuples of different kinds can be quite useful when specifying systems. Although as a discipline it does not enhance the expressiveness of the calculus, it can significantly simplify system descriptions by enabling the use of a single, multi-purpose name (which can be used in many ways) instead of employing several names to perform distinct kinds of communication. Indeed, the ability of exploiting typing information to give subtler and simpler analyses for systems is an important factor in our choice of typing system. The well-typed process expressions and abstractions are those which may be assigned an agent type as follows:

- $\Sigma_{i \in I} \alpha_i. P_i : \text{abs}()$, if for each $i \in I$, $P_i : \text{abs}()$ and
 - α_i is of the form $\bar{t}(\tilde{t})$ where $t : L$, $\tilde{t} : \tilde{i}$ and $\tilde{i} \in \lambda(L)$, or
 - α_i is of the form $t(\tilde{u})$ where $t : L$, $\tilde{u} : \tilde{i}$ and $\tilde{i} \in \lambda(L)$, or
 - $\alpha_i = \tau$.
- $P \mid Q : \text{abs}()$, if $P : \text{abs}()$ and $Q : \text{abs}()$.
- $(\nu x)P : \text{abs}()$, if $P : \text{abs}()$.
- $\text{cond}(t_1 \triangleright P_1, \dots, t_n \triangleright P_n) : \text{abs}()$, if for each i , $P_i : \text{abs}()$ and $t_i : \text{bool}$.
- $F(\tilde{t}) : \text{abs}()$, if $F : \text{abs}(\tilde{i})$ and $\tilde{t} : \tilde{i}$.
- $(\tilde{u})P : \text{abs}(\tilde{i})$, if $\tilde{u} : \tilde{i}$ and $P : \text{abs}()$.
- $D : \xi$, if D is of type ξ .

It is easy to see that our type system is, in general, not sound, in the sense that a well-typed process is not necessarily free from mismatches. For example, suppose $a : A$ with $\lambda(A) = \{(A), (A, A)\}$ and let

$$P = a(x).0 \mid \bar{a}(b, c).0.$$

Then although P is well-typed, it presents a mismatch. This is not surprising as we have imposed a type discipline which allows names to be used in more than one way. Nonetheless, we might expect that mismatch freedom would be ensured if we could guarantee that whenever an input prefix $t(\tilde{u})$ occurs within a process it does so within a summation accompanied by summands with prefixes corresponding to all possible inputs via t allowed by the sorting. For example,

$$Q = (a(x).0 + a(x, y).0) \mid \bar{a}(b, c).0$$

presents no mismatch unlike P above. So consider the sublanguage, Pro , generated by the following agent expressions:

$$\begin{aligned} P &::= \sum_{i \in I} \alpha_i. P_i \mid P \mid Q \mid (\nu x)P \mid F(\tilde{t}) \mid \text{cond}(t_1 \triangleright P_1, \dots, t_n \triangleright P_n) \\ F &::= (\tilde{u})P \mid D \end{aligned}$$

where if $\alpha_j = t(\tilde{u})$, $t : i$, $\lambda(i) = \{\tilde{i}_1, \dots, \tilde{i}_n\}$ then there exist $j_1, \dots, j_n \in I$ such that $\alpha_{j_k} = t(\tilde{u}_{j_k})$ and $\tilde{u}_{j_k} : \tilde{i}_k$, for $1 \leq k \leq n$. We say that a type system is *sound* for a language if each of its well-typed terms is free of mismatches. We have the following:

Theorem 2.2.1 The type system is sound for Pr_0 .

PROOF: The proof is based on the following ideas: If $P \in \text{Pr}_0$ and P is well-typed then it is easy to see that it is immediately free of mismatches, that is, if it contains unguarded the prefix $\bar{a}(\tilde{u})$ then either a does not occur unguarded in positive subject position in P or it occurs unguarded in a prefix of the form $a(\tilde{u}')$ where, if $\tilde{u} : \tilde{T}$, then $\tilde{u}' : \tilde{T}$. Furthermore, if $P \xrightarrow{\mu} P'$ (for the transition system defined in the following section) then P' is well-typed and $P' \in \text{Pr}_0$. \square

It is not our intention to study this type system in more detail; this result is sufficient for our purposes. As we will see, all applications we study in the remainder of the thesis consider processes of Pr_0 . Thus, assuming that they are well-sorted, we may be sure that they are free of mismatches.

2.3 Labelled transition semantics

The operational semantics for the π_v -calculus is given as a labelled transition system

$$(\text{Pr}, \{\xrightarrow{\mu} \mid \mu \in \text{Act}^+\}),$$

where Act^+ is the set of *labels* or *actions*. The system we consider has *early instantiation* of names: the bound names of an input are instantiated in the rule for input. This is in contrast to a late transition system where input instantiations only take place in the rule for communication, (for an example of such a system see [MPW92]). We choose to work with this system for two main reasons. First, the bisimulation supported in this setting, *early congruence*, coincides with reduction (or barbed) congruence [San92]. Secondly, the structure of the transitions allows a much clearer presentation of the notions of determinacy and confluence discussed later in the thesis, and the theory developed is simpler than if the late setting were chosen instead.

The transition system is obtained from that of the π -calculus by incorporating a treatment of terms. The set of actions, $\text{Act}^+ = \{\tau\} \cup \text{Act}$, where τ is the silent action representing interaction and Act is the set of visible actions. There are two types of visible actions, where x, \tilde{y} are names and \tilde{v} are values:

- $(\nu \tilde{y})\bar{x}(\tilde{v})$ the output action; it is assumed that $\tilde{y} \subseteq \tilde{v}$ and the action represents the creation of new names \tilde{y} and output of the tuple \tilde{v} via name x .
- $x(\tilde{v})$ the input action; the tuple \tilde{v} is received via name x .

Note that the actions above must satisfy the sorting of the calculus: if $x : L$ and $\tilde{v} : \tilde{i}$ then it must be that $\tilde{i} \in \lambda(L)$. As was the case for input and output prefixes, in

both cases we refer to x as the *subject* and \tilde{v} as the *object* of the action. Moreover, we say that in the input action, x occurs in *positive* subject position (or is a *positive* subject), whereas in the output action it occurs in *negative* subject position (or is a *negative* subject). We use the following notation:

Notation 2.3.1 We write $\text{subj}(\mu)$ for the subject and $\text{obj}(\mu)$ for the object of an action $\mu \in \text{Act}$. If $s = \mu_1 \dots \mu_n$, where $\mu_i \in \text{Act}$ for all i we write $\text{subj}(s)$ for the sequence $\text{subj}(\mu_1) \dots \text{subj}(\mu_n)$. Finally, for L a sort we write L^+ (resp. L^-) for the set of input actions with a positive (resp. negative) subject whose name is of sort L , and L^\pm for $L^+ \cup L^-$.

An additional point to note is the difference between the brackets used in input prefix (round brackets) and those used in the input action (angled brackets). This is to emphasize the fact that in the input prefix $x(\tilde{u})$, \tilde{u} are variables waiting to be instantiated, whereas in the input action $x\langle\tilde{v}\rangle$, \tilde{v} represent values, symmetrically to the values in an output $\bar{x}\langle\tilde{v}\rangle$. Given an action μ , the bound and free names of μ are given as follows:

| μ | $\text{fn}(\mu)$ | $\text{bn}(\mu)$ |
|--|-----------------------------------|------------------|
| τ | \emptyset | \emptyset |
| $x\langle\tilde{v}\rangle$ | $x \cup n(\tilde{v})$ | \emptyset |
| $(\nu\tilde{y})\bar{x}\langle\tilde{v}\rangle$ | $x \cup n(\tilde{v}) - \tilde{y}$ | \tilde{y} |

We let $n(\mu) = \text{bn}(\mu) \cup \text{fn}(\mu)$ to denote the names of μ . As with processes, we use α -conversion on actions and identify actions ρ, ρ' , and write $\rho = \rho'$, that differ only by a change of bound names. Further, we always assume that the bound names of output actions are *fresh*. That is they do not occur in agents performing the action. We employ the following notation.

Notation 2.3.2 We write $\alpha \text{ comp } \beta$ if $\alpha = x\langle\tilde{v}\rangle$ and $\beta = (\nu\tilde{y})\bar{x}\langle\tilde{v}\rangle$ or vice versa. If $\alpha \text{ comp } \beta$ we say that α is *complementary* to β .

The transition system is presented in Table 2.1. Note that we have omitted the symmetric versions of rules (PAR) and (COM).

First consider rule (ALPHA). Its use allows us to define transitions up to α -conversion, that is if two agents are only different by a change of bound names and variables, then they have the same transitions. Thus, some side-conditions in the transition system may be avoided. The first four rules are straightforward. In the rule (PAR) the side condition prevents transitions of the form

$$(\nu x)\bar{y}\langle x \rangle. x(y) \mid \bar{x}\langle z \rangle \xrightarrow{(\nu x)\bar{y}\langle x \rangle} x(y) \mid \bar{x}\langle z \rangle.$$

| | | |
|---------|--|--|
| (IN) | $t(\tilde{u}).P \xrightarrow{x(\tilde{v})} P\{\tilde{v}/\tilde{u}\}$ | provided $t \rightarrow x$, and if $\tilde{u} : \tilde{i}$ then $\tilde{v} : \tilde{i}$ |
| (OUT) | $\bar{t}(\tilde{t}).P \xrightarrow{\bar{x}(\tilde{v})} P$ | provided $t \rightarrow x, \tilde{t} \rightarrow \tilde{v}$ |
| (TAU) | $\tau.P \xrightarrow{\tau} P$ | |
| (SUM) | $\frac{P_j \xrightarrow{\mu} P'_j}{\Sigma_{i \in I} P_i \xrightarrow{\mu} P'_j}$ | provided $j \in I$ |
| (PAR) | $\frac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q}$ | provided $\text{bn}(\mu) \cap \text{fn}(Q) = \emptyset$ |
| (COM) | $\frac{P \xrightarrow{x(\tilde{v})} P' \quad Q \xrightarrow{(\nu \tilde{y})\bar{x}(\tilde{v})} Q'}{P \mid Q \xrightarrow{(\nu \tilde{y})} (P' \mid Q')}$ | provided $\tilde{y} \cap \text{fn}(P) = \emptyset$ |
| (RES) | $\frac{P \xrightarrow{\mu} P'}{(\nu x)P \xrightarrow{\mu} (\nu x)P'}$ | provided $x \notin \text{n}(\mu)$ |
| (OPEN) | $\frac{P \xrightarrow{(\nu \tilde{y})\bar{x}(\tilde{v})} P'}{(\nu z)P \xrightarrow{(\nu z \tilde{y})\bar{x}(\tilde{v})} P'}$ | provided $z \in \tilde{v} - (\tilde{y} \cup x)$ |
| (COND) | $\frac{P_i \xrightarrow{\mu} P'_i}{\text{cond}(b_1 \triangleright P_1 \dots b_n \triangleright P_n) \xrightarrow{\mu} P'_i}$ | if $b_i \rightarrow \text{true}, \forall j < i : b_j \rightarrow \text{false}$ |
| (CONST) | $\frac{P\{\tilde{v}/\tilde{u}\} \xrightarrow{\mu} P'}{D(\tilde{v}) \xrightarrow{\mu} P'}$ | $D \stackrel{\text{def}}{=} (\tilde{u})P$ |
| (ALPHA) | $\frac{P \xrightarrow{\mu} P', P = Q}{Q \xrightarrow{\mu} P'}$ | |

Table 2.1: Labelled transition system for Pr

This is required as the name x in the first component is meant to be a fresh name not to be confused with the free name of the second component. However, by using α -conversion, since $(\nu x)\bar{y}\langle x \rangle. x(y) = (\nu w)\bar{y}\langle w \rangle. w(y)$, the following transition is possible.

$$(\nu x)\bar{y}\langle x \rangle. x(y) \mid \bar{x}\langle z \rangle \xrightarrow{(\nu w)\bar{y}\langle w \rangle} w(y) \mid \bar{x}\langle z \rangle$$

A similar purpose is served by the side-condition of (COM). To understand the intuition behind the side-condition of (RES) consider the following transition:

$$(\nu y)a(x). P \xrightarrow{a(y)} (\nu y)P\{y/x\}$$

The static binding of y assumed by the restriction is violated by the transition. Note however, that this does not imply that name y may not be received by the agent. To achieve this, it is necessary to α -convert the bound name required by the process to a fresh name:

$$(\nu y)a(x). P = (\nu z)a(x). P\{z/y\} \xrightarrow{a(y)} (\nu z)P\{y/x\}$$

Rule (OPEN) implements scope extrusion: such communications allow private names to be emitted by a process and thus carried out of their current scopes. Finally, note that (COND) makes precise the left to right evaluation of the guards in the conditional operator and the choice of the agent corresponding to the first guard evaluating to true.

We conclude by recalling some standard notation. As usual we say that Q is a *derivative* of P if there are $n \geq 0$, μ_1, \dots, μ_n such that $P \xrightarrow{\mu_1} \dots \xrightarrow{\mu_n} Q$. Moreover, given $\mu \in Act^+$, $s \in Act^*$, $s = \mu_1 \dots \mu_n$ we write

$$\begin{aligned} P &\Rightarrow Q && \text{iff} && P \xrightarrow{\tau}^* Q \\ P &\xRightarrow{\mu} Q && \text{iff} && P \Rightarrow \xrightarrow{\mu} \Rightarrow Q \\ P &\xRightarrow{s} Q && \text{iff} && \text{there exist } P_1 \dots P_{n-1} \text{ such that } P \xRightarrow{\mu_1} P_1 \xRightarrow{\mu_2} \dots P_{n-1} \xRightarrow{\mu_n} Q \\ P &\xrightarrow{\hat{\mu}} Q && \text{iff} && P \xrightarrow{\mu} Q, \text{ or } \mu = \tau \text{ and } P = Q \\ P &\xRightarrow{\hat{\mu}} Q && \text{iff} && P \Rightarrow \xrightarrow{\hat{\mu}} \Rightarrow Q. \end{aligned}$$

Finally, we say that a set S of agents is *derivation-closed* if, whenever $P \in S$ and $P \xrightarrow{\mu} Q$, then $Q \in S$.

2.4 Behavioural equivalence

Observational equivalence is based on the idea that two equivalent systems exhibit the same behaviour at their interfaces with the environment. This requirement was

captured formally through the notion of *bisimulation*, originally suggested by Park [Par81] and Milner [Mil80], which is a binary relation on the states of systems. Two states are bisimilar if for each single computational step of the one there exists an appropriate matching (multiple) step of the other, leading to bisimilar states. A variety of bisimulations (as well as other formalizations of behavioural equivalence) have emerged over recent years in the search for notions which are mathematically tractable, reasonable and appropriate for different types of applications, where the criterion for indistinguishability of systems may vary.

This section is devoted to presenting some well-established theory regarding the notion of (bi)simulation, which we use in the thesis. We begin by presenting the notion of strong simulation and bisimulation for the π_v -calculus.

Definition 2.4.1 A binary relation \mathcal{R} is a *strong simulation* if PRQ implies that for all $\alpha \in Act^+$ with $\text{bn}(\alpha) \cap \text{fn}(P, Q) = \emptyset$,

if $P \xrightarrow{\alpha} P'$ then for some $Q', Q \xrightarrow{\alpha} Q'$ and $P'\mathcal{R}Q'$.

The relation \mathcal{R} is a *strong bisimulation* if both \mathcal{R} and \mathcal{R}^{-1} are strong simulations. *Bisimilarity*, \sim , is the largest strong bisimulation.

Thus, strong bisimulation treats all actions uniformly. The notion of bisimulation that abstracts away from silent actions, referred to as *weak bisimulation*, is given below:

Definition 2.4.2 A binary relation \mathcal{R} is a *weak simulation* if PRQ implies that for all $\alpha \in Act^+$ with $\text{bn}(\alpha) \cap \text{fn}(P, Q) = \emptyset$,

if $P \xrightarrow{\alpha} P'$ then for some $Q', Q \xRightarrow{\hat{\alpha}} Q'$ and $P'\mathcal{R}Q'$.

The relation \mathcal{R} is a *weak bisimulation* if both \mathcal{R} and \mathcal{R}^{-1} are weak simulations. *Weak bisimilarity*, \approx , is the largest weak bisimulation.

Each of \sim and \approx is an equivalence. However, neither of them is a congruence as they are not preserved by input prefix. To obtain the full congruence, it is sufficient to require bisimilarity over all substitutions of names ([MPW92]). For example, letting \approx be the symbol of weak congruence, we define

$$P \approx Q \quad \text{iff for all } \sigma \quad P\sigma \approx Q\sigma.$$

Note that the notions we have presented have appeared in the literature under the names of *early* strong bisimilarity, *early* weak bisimilarity and *early* congruence and have been differentiated from the respective *late* notions. Informally,

the difference between the two settings arises in the instantiation of the names of an input: in the early case this occurs at the instant the input action is inferred. By contrast, in the late case names are instantiated at the moment a communication is inferred. In other words, in the early setting an input is considered to be a single atomic event, whereas in the late setting it is considered as two: the commitment on the use of a channel for communication and then the receipt of an object. This results in two distinct notions with early bisimilarity being the weaker of the two. For a discussion, see [MPW92, MPW93].

In the thesis we will only use early bisimulation and thus we will simply refer to it as ‘bisimulation’. We do this for a number of reasons. First, early bisimulation allows a simpler definition by treating input and output actions symmetrically, unlike late bisimulation. Moreover, checks for the existence of a bisimulation do not require quantification over names, which are imposed by the strong version. Finally, early congruence coincides with the notion of barbed, or reduction congruence [San92], a natural notion defined uniformly in process calculi and based on reduction semantics.

We proceed to consider two variants of weak bisimulation. The former is the stronger notion of branching bisimilarity of [GW89] and the latter is the weaker notion of coupled simulation of [PS92]. We conclude by refining the notion of bisimulation to handle divergent behaviour.

2.4.1 Branching bisimilarity

Observational equivalences are often distinguished between those satisfying a *linear time* semantics, and those satisfying a *branching time* semantics. In the former setting a process is completely determined by the observable content of its runs whereas in the latter information about the structure of the process and, in particular, points where paths diverge is also preserved.

It is commonly accepted that bisimulation equivalence respects branching time for processes without silent actions. In fact, for this setting an equivalence respects branching time if it is at least as demanding as bisimulation equivalence. In the presence of silent actions, however, weak bisimilarity fails to respect branching time. For example, consider the transition systems of Figure 2.1.

It is easy to see that they are weakly bisimilar. However, consider the dotted path of the first system involving the run ab . This path does not pass through a point where the action c is possible. On the other hand, the second system is such that every path with visible content ab must pass through a state where c is possible and moreover, unlike the first system, it passes through a state where the action

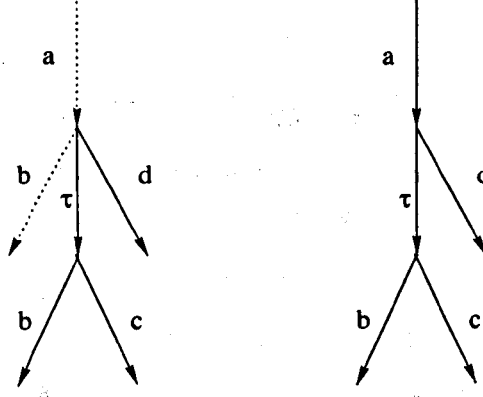


Figure 2.1: $a.(b + \tau.(b + c) + d)$ vs. $a.(\tau.(b + c) + d)$

d is no longer possible. This strongly suggests that the two systems should not be identified by a branching time equivalence.

This resulted in the notion of *branching bisimulation* proposed in [GW89] where the example above originates. In the same work it was argued that while branching bisimulation coincides with ordinary bisimulation in the absence of silent actions it also takes over its rôle as a characterization of branching time equivalence in the general setting. For a discussion of the significance of branching time semantics and a formalization of the concepts involved see [Gla93].

The definition of branching bisimulation in the π_v -calculus is as follows:

Definition 2.4.3 A relation \mathcal{B} is a *branching simulation* if $P\mathcal{B}Q$ implies that for all $\alpha \in Act^+$ with $\text{bn}(\alpha) \cap \text{fn}(P, Q) = \emptyset$, if $P \xrightarrow{\alpha} P'$, then either

1. $\alpha = \tau$ and $P'\mathcal{B}Q$, or
2. $Q \Rightarrow Q'' \xrightarrow{\alpha} Q'$ for some Q'', Q' such that $P\mathcal{B}Q''$ and $P'\mathcal{B}Q'$.

The relation \mathcal{B} is a *branching bisimulation* if both \mathcal{B} and \mathcal{B}^{-1} are branching simulations. *Branching bisimilarity*, \simeq , is the largest branching bisimulation.

We recall that

- \simeq is an equivalence, and
- $\simeq \subset \approx$.

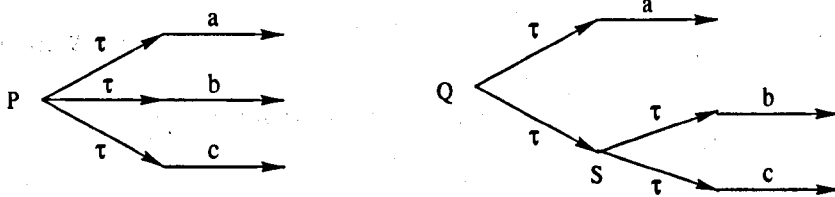
Branching bisimilarity is not a congruence. Branching congruence, which we denote by \simeq , is defined by setting $P \simeq Q$ iff $P\sigma \simeq Q\sigma$ for all substitutions σ .

2.4.2 Coupled simulations

Consider the agents

$$\begin{aligned} P &\stackrel{\text{def}}{=} \tau.a + \tau.b + \tau.c \\ Q &\stackrel{\text{def}}{=} \tau.a + \tau.(\tau.b + \tau.c) \end{aligned}$$

and their transition graphs.



It is easy to check that the two agents are not bisimilar. The point of interest is state S in the transition system of Q . Clearly, at S there is the possibility of performing a run with visible content b or c . However, there is no such state in the transition system of P , where the choice of performing one of a , b or c is resolved with the occurrence of the first (silent) action. Consequently, there exists no bisimulation relating the two agents. Nonetheless there are many applications in which one would want to consider agents such as P and Q as equivalent. Thus there has been considerable effort to define equivalence notions that equate systems in which internal choices are not necessarily resolved simultaneously.

A resulting notion following the simulation model is that of *coupled simulation* due to Parrow and Sjodin, [PS92, PS94] (for related notions based on the testing model, and associated with testing equivalence, see [Hen88, Gla93]). While weak bisimilarity requires that agents *bisimulate* each other at all states, coupled simulation requires that two equivalent systems may simulate each other and bisimulate each other only at stable states, that is states where no silent actions are possible. Thus it requires the existence of two simulation relations between the agents in question (one ‘in each direction’) which coincide, or *are coupled*, at stable states.

Definition 2.4.4 An agent P is *stable* if whenever $P \xrightarrow{\alpha} P'$ then $\alpha \neq \tau$. Otherwise, P is *unstable*.

A coupled simulation is defined as follows:

Definition 2.4.5 The pair (S_1, S_2) is a *coupled simulation* if S_1 and S_2^{-1} are (weak) simulations satisfying the following

- If PS_1Q and P is stable, then PS_2Q .
- If PS_2Q and Q is stable, then PS_1Q .

Two agents P and Q are *cs-equivalent*, $P =_{cs} Q$, if they are related by both components of a coupled simulation.

As shown in [PS94], $=_{cs}$ is an equivalence for convergent agents. Moreover, it has been shown to be weaker than bisimulation and stronger than testing equivalence.

Proposition 2.4.6

1. $=_{cs}$ is an equivalence for convergent processes.
2. $\approx \subseteq =_{cs}$.

In order to generalize the notion of coupled simulation to divergent processes it appears that one loses one of the original's advantages, namely a characterization over single transitions. The notion proposed in [PS94] is the following:

Definition 2.4.7 The pair (S_1, S_2) is a *weak coupled simulation* if S_1 and S_2^{-1} are (weak) simulations satisfying the following

- If PS_1Q then there exists Q' such that $Q \Rightarrow Q'$ and PS_2Q' .
- If PS_2Q then there exists P' such that $P \Rightarrow P'$ and $P'S_1Q$.

Two agents P and Q are *cw-equivalent*, $P =_{cw} Q$, if they are related by both components of a weak coupled simulation.

It is easy to see that $=_{cw}$ is an equivalence and it coincides with $=_{cs}$ for convergent agents.

Proposition 2.4.8

1. $=_{cw}$ is an equivalence relation.
2. $=_{cs} = =_{cw}$ for convergent agents.

For more details concerning these notions see [PS94, Gla93].

2.4.3 Divergence

One feature of the relations we have considered above, noted and criticized by Milner in [Mil89], concerns the (lack of) treatment of the phenomenon of divergence: it is possible for two processes, exactly one of which is divergent, to be related to each other. This property may be useful in many applications but it is inappropriate in others where it is crucial to take account of the presence of infinite silent paths. This subject has been investigated in [Mil81, HP80, Abr87, Wal90]. Here we simply record some notation we will employ in the thesis and two notions of bisimulation sensitive to divergence appropriate for our purposes.

We begin with some terminology.

Notation 2.4.9 We say that an agent P *diverges*, written $P \uparrow$, if P can perform an infinite sequence of τ actions; otherwise P *converges*, $P \downarrow$.

We also employ the following.

Definition 2.4.10

- $P \downarrow \tau$ if $P \downarrow$;
- $P \downarrow (\nu \tilde{w}) \bar{x}(\tilde{v})$, if $P \downarrow$ and whenever $P \xrightarrow{(\nu \tilde{w}) \bar{x}(\tilde{v})} P'$ then $P' \downarrow$;
- $P \downarrow x(\tilde{v})$, if $P \downarrow$ and for all \tilde{v}' whenever $P \xrightarrow{x(\tilde{v}')} P'$ then $P' \downarrow$.

We read $P \downarrow \alpha$ as P converges on α . According to the definition, an agent converges on an action if it converges and remains convergent after performing that action. Note that there is an asymmetry between the treatment of input and output actions: the clause for inputs involves a quantification over all possible inputs performed via name x which is not required for output actions as the values output are determined by the action. The intention of this definition is to ensure that, no matter what values are received in the action, the resulting agent will be convergent. So, for example, if $P \equiv x(z). \text{cond}(z = 0 \triangleright \Omega, \text{true} \triangleright 0)$ then $\text{not}(P \downarrow x(v))$, since $P \xrightarrow{x(0)} \Omega \uparrow$. An alternative formalization of this concept would be to consider divergence on names as opposed to actions. However, we opt for the notation above as it turns out to be more convenient when reasoning about convergence for a sequence of actions. Moreover, we write the following:

Notation 2.4.11 Letting s range over Act^* we write $P \downarrow s$ if whenever $P \xrightarrow{t} P'$ with t a prefix of s , $P' \downarrow$. We let $\uparrow \alpha$ and $\uparrow s$ be the complementary relations to $\downarrow \alpha$ and $\downarrow s$ respectively.

The definition of divergence-sensitive bisimilarity follows.

Definition 2.4.12 The relation *d-bisimilarity*, \approx_1 , is the largest such that if $P \approx_1 Q$ then for all $\alpha \in Act^+$ with $\text{bn}(\alpha) \cap \text{fn}(P, Q) = \emptyset$,

1. if $P \downarrow \alpha$ then $Q \downarrow \alpha$ and if $P \xrightarrow{\alpha} P'$ then $Q \xRightarrow{\hat{\alpha}} Q'$ and $P' \approx_1 Q'$, and
2. vice versa.

Thus, d-bisimilarity relates agents that may match each other's actions for as long as computation does not introduce divergence. Moreover, two d-bisimilar agents are convergent on exactly the same set of actions.

We also consider a divergence-sensitive variant of branching bisimilarity.

Definition 2.4.13 The relation *db-bisimilarity*, \simeq_1 , is the largest such that if $P \simeq_1 Q$ then for all $\alpha \in Act^+$ with $\text{bn}(\alpha) \cap \text{fn}(P, Q) = \emptyset$,

1. if $P \downarrow \alpha$ then $Q \downarrow \alpha$ and if $P \xrightarrow{\alpha} P'$ then
 - (a) $Q \Rightarrow Q'' \xrightarrow{\alpha} Q'$ with $P \simeq_1 Q''$ and $P' \simeq_1 Q'$, or
 - (b) $\alpha = \tau$ and $P' \simeq_1 Q$, and
2. vice versa.

Hence, db-bisimilarity adopts the same approach to divergent behaviour as d-bisimilarity while it requires agents to match each other's actions in the branching bisimilarity fashion. We record the following results:

Proposition 2.4.14

1. \approx_1 and \simeq_1 are equivalences.
2. $\simeq_1 \subset \approx_1$.

However, note that \approx_1 and \simeq_1 are unrelated to \approx and \simeq respectively. For example, $\alpha + \Omega \simeq_1 \beta + \Omega$ though $\alpha + \Omega \not\approx \beta + \Omega$. On the other hand, $\alpha + \alpha.\Omega \simeq \alpha$ whereas $\alpha + \alpha.\Omega \not\approx_1 \alpha$. It is not surprising however, that the notions coincide for systems that do not exhibit divergent behaviour.

Definition 2.4.15 An agent P is *fully convergent* if $P \downarrow$ and for all derivatives P' of P , $P' \downarrow$.

Proposition 2.4.16 $\approx_1 = \approx$ and $\simeq_1 = \simeq$ for fully convergent agents.

PROOF: Straightforward by the definitions. □

2.5 Friendly agents

We conclude this chapter by introducing some useful properties of agents. We consider these in the context of a type of agents which have a close affinity with the *friendly systems* considered in [Mil92b]. These are built from the special kind of agents defined below.

Definition 2.5.1 An agent S is *consistent* if for all derivatives P of S , whenever $P \xrightarrow{\alpha}$ and $P \xrightarrow{\beta}$, where α and β are both output actions with $\text{subj}(\alpha) = \text{subj}(\beta)$, then $\alpha = \beta$.

Let

$$S \equiv (\nu \tilde{x})(P_1 \mid \dots \mid P_l \mid C_1 \mid \dots \mid C_m \mid !C_{m+1} \mid \dots \mid !C_n)$$

We call each P_i , C_j and $!C_k$ a *component* of S and we say that S is a *friendly agent* if it satisfies the following conditions:

1. each P_i is a derivative of some C_j ,
2. for all $m+1 \leq j \leq n$, $C_j \equiv (\nu \tilde{y})\alpha_j.Q_j$, for some α_j , Q_j , and
3. each C_j is consistent. □

Note that although we have used the same adjective our friendly *agents* differ from the friendly *systems* of [Mil92b]. In particular we allow parallel composition and agent constants to occur within a C_j . However, we require that each C_j is a consistent agent, that is at any point it is capable of transmitting at most one value via any name, although a friendly agent itself need not be consistent. This condition is a reasonable one to impose for the kind of friendly agents we wish to consider below. We observe that the form of a friendly agent is preserved under derivation although the number of the P components may change as it is possible that the system spins off copies of its replicators. We employ the following definitions.

Definition 2.5.2 An agent P *bears* a name x , or x is *borne* in P , if x occurs free in P in a positive subject position. Further, P *handles* x , or x is *handled* in P , if x occurs free in P in a negative subject position.

Thus, an agent bears a name x if, syntactically, it has the capability of receiving input via x . Dually, an agent handles a name if, syntactically, it may use it for output. We find it useful to isolate a further property regarding the use of names that generalizes that of handling, namely the capability of either handling a name, or sending it as an object of an output action. Thus we have the following definition.

Definition 2.5.3 An agent P *controls* a name x , or x is *controlled* in P , if x appears in P free in negative subject position or in object position.

We continue with a further definition that refines the notions of bearing, handling and controlling.

Definition 2.5.4 A friendly agent S *uniquely bears* (resp. *handles*, *controls*) a name x , or x is *uniquely borne* (resp. *handled*, *controlled*) in S if for each derivative S' of S , x is borne (resp. *handled*, *controlled*) by at most one component of S' .

Intuitively this definition implies that if x is uniquely borne in S then a communication via x may be received by at most one component of S . Similarly, if x is uniquely handled then at most one component may send via x at any time. In addition, if x is uniquely controlled then not only may at most one component send via x but that component is also solely responsible for who may subsequently do so.

In [Mil92b], the properties of unique bearing and unique handling were investigated and conditions were identified which guarantee that they are satisfied by a friendly system (as defined in that context), and the importance of the properties was discussed. It turns out that a similar kind of property is relevant in the context of this work. In particular it is useful for guaranteeing determinacy of agents as we will see in Chapter 3. The property required is given in the following definition.

Definition 2.5.5 A friendly agent S *manages* a name x if, either

- $x \in \text{bn}(S)$ and x is borne by at most one component of S and controlled by at most one component of S , or
- $x \in \text{fn}(S)$ and either x is borne by at most one component of S , or x is controlled by at most one component of S but not both.

Moreover, we say that S *persistently manages* x if for all derivatives S' of S , S' manages x .

Thus, if a name is managed by an agent then it is either internal to the system and it can be used for communication between at most two components, or it is free and it can be used by at most one component for either sending or receiving. Further, a name is persistently managed by an agent if it is managed throughout computation.

We want to enunciate conditions on friendly agents that guarantee the persistent management of names. So, let $S \equiv (\nu \tilde{x})(\Pi P_i \mid \Pi C_j \mid \Pi! C_k)$ be a friendly agent where $C_k = (\nu \tilde{y})\alpha_j.Q_j$ and $\text{subj}(\alpha_j) : L_j$. Moreover, let x be a name of sort $A \neq L_j$ and suppose S manages x . It is easy to see that the property of managing

a name is not preserved under derivation. For one thing, a replicator may produce two components both bearing or controlling x . Thus we require the following:

- (i) for all k , $x \notin \text{fn}(C_k)$.

In addition, a component may acquire the bearing of a name. For example

$$v(z).z(w).0 \mid \bar{v}(x).x(z).0 \longrightarrow x(w).0 \mid x(z).0$$

Similarly, a component may acquire the capability of handling a name. For reasons that will become apparent in our later study of concurrent-object programs we take the following approach for preventing such situations from arising: first, we require that the ability of bearing may not be acquired by a component. That is when a component P receives a name of sort A , it is not allowed to subsequently bear it. Secondly, the capability of controlling may be freely passed between components, but, to ensure unique controlling, when a component transmits a name of sort A , it may not transmit or handle it again (unless and until it receives it again). This leads to the following conditions:

- (ii) if P is a derivative of a C_j , $P \xrightarrow{\alpha} P'$, $\alpha = a(\dots, y, \dots)$, $y : A$ then y is not borne in P' ;
- (iii) if P is a derivative of a C_j , $P \xrightarrow{\alpha} P'$, $\alpha = (\nu \tilde{w})\bar{a}(\dots, y, \dots)$, $y : A$ then y is not controlled in P' .

Although very close to our goal of ensuring the persistent management of x , there is a further point to consider. Let $P_1 \equiv a(y).\bar{y}(z).0$, $P_2 \equiv \bar{x}(w).0$ and consider $S \equiv P_1 \mid P_2$. We can see that $x \in \text{fn}(S)$ and at most one component of S , P_2 , controls x . However, consider the following transition:

$$S \xrightarrow{a(x)} S' = \bar{x}(z).0 \mid \bar{x}(w).0$$

Clearly, S' does not manage x since it is controlled by both of its components. The cause of this was the receipt of name x which was known to S (note that if x did not occur in S , its receipt would not violate the persistent management property). A possible solution is to enforce restrictions on the free names of S , requiring that they cannot be used to receive names of sort A . The following definition is useful for formalizing this requirement.

Definition 2.5.6 Let $\tilde{I} = I_1 \dots I_n$ be a sequence of sorts. An agent P is \tilde{I} -closed if none of its derivatives can perform an action in any I_j^\pm .

Thus an agent is \tilde{I} -closed if no name of sort \tilde{I} may appear at the interface of the agent with the environment. A sufficient condition to ensure that an agent is closed with respect to a tuple of sorts is given in the following.

Lemma 2.5.7 Let P be an agent such that $\text{fn}(P)[\tilde{I} = \emptyset]$ respecting the sorting λ where, if $I_j \in \bigcup \lambda(I')$ then $I' \in \tilde{I}$. Then P is \tilde{I} -closed.

PROOF: According to the sorting no name of sort $J \notin \tilde{I}$ may carry \tilde{I} -names. Thus, since no name of a sort in \tilde{I} occurs free in P and P satisfies λ , $\text{fn}(P')[\tilde{I} = \emptyset]$, for all derivatives P' of P , and so P is \tilde{I} -closed. \square

We have the following result.

Proposition 2.5.8 Let S be a friendly agent and $x : A$ a name such that S manages x and S satisfies (i)-(iii) above, for x and A . Let λ be a sorting and \tilde{I} a set of sorts such that if $A \in \bigcup \lambda(L)$ then $L \in \tilde{I}$. Suppose additionally that S is well-sorted according to λ and \tilde{I} -closed. Then S persistently manages x .

In words, the side-condition on free names requires that if y is a free name of a derivative of S then its sorting does not allow it to carry names of sort A . Therefore, the name x may not be received (or transmitted) by the process, although it may be used as a channel for communication with the environment, if, that is, it occurs free.

PROOF: We want to prove that for all derivatives S' of S , S' manages x . The proof follows by induction on the length of the transition $S \xRightarrow{*} S'$ and the significance of the assumptions is clearly illustrated. So suppose $S \xRightarrow{*} S'' \xrightarrow{\alpha} S'$, where $S'' = (\nu \tilde{p})(\Pi P_i \mid !C_j)$, and S'' manages x . There exist two possibilities for the transition $S'' \xrightarrow{\alpha} S'$.

- $\alpha = \tau$ and $S' = (\nu \tilde{p}')(P'_1 \mid P'_2 \mid \dots)$ where $P_1 \xrightarrow{\beta} P'_1$ and $P_2 \xrightarrow{\bar{\beta}} P'_2$, $\beta \text{ comp } \bar{\beta}$. If $x \notin (\text{fn}(\beta) \cup \text{fn}(\bar{\beta}))$ then by induction the claim follows. Similarly, if $x \notin (\text{obj}(\beta) \cup \text{obj}(\bar{\beta}))$, since no component has acquired the ownership of x , by induction x is managed in S' . Finally, if $x \in (\text{obj}(\beta) \cup \text{obj}(\bar{\beta}))$ then by (ii) and (iii) we may conclude that the claim holds.
- $\alpha \neq \tau$ and either $C_1 \xrightarrow{\alpha} P$, $S' = (\nu \tilde{p}')(\Pi P_i \mid P \mid !C_j)$, or $P_1 \xrightarrow{\alpha} P'_1$ and $S' = (\nu \tilde{p}')(P'_1 \mid \Pi_{i \neq 1} P_i \mid !C_j)$. We note that by the assumption of the proposition, $x \notin \text{obj}(\alpha)$. Thus no component of S'' has acquired the capability of using x and so, x is managed in S' .

This completes the proof. \square

As a special case of the result above suppose S is such that $\text{fn}(S) = \emptyset$. It is easy to see that $\text{fn}(S') = \emptyset$ for all derivatives S' of S and thus S persistently manages x assuming it satisfies the remaining properties.

An alternative solution to ensure that communication with the environment will not affect the persistent management of a name is to require that the environment will be 'well-behaved' in some sense. If so we may then guarantee that the management of name x will not be violated. The assumption we make is that the environment will not provide a derivative S' of S with name x unless $x \notin \text{fn}(S')$. So, let $|S|$ be the set of the derivatives of S obtained after a sequence of actions that satisfy this assumption:

$$|S| = \{P \mid \exists n, \mu_1 \dots \mu_n : S = S_0 \xrightarrow{\mu_1} S_1 \xrightarrow{\mu_2} \dots \xrightarrow{\mu_n} S_n = P, \\ \text{and, if } \mu_i = a(\tilde{y}) \text{ and } x \in \tilde{y}, \text{ then } x \notin \text{fn}(S_{i-1})\}$$

We have the following result:

Proposition 2.5.9 Let S be a friendly agent and $x : A$ a name such that S manages x and S satisfies (i)-(iii) above for x and A . Then for every S' in $|S|$, S' manages x .

PROOF: The proof is by induction on the length of the derivation $S \xrightarrow{*} S'$, where $S' \in |S|$. \square

We generalize the notion of managing a name as follows:

Definition 2.5.10 Let $S = (\nu \tilde{x})(\Pi P_i \mid \Pi! C_k)$ be a friendly agent where $C_j \equiv (\nu \tilde{y}_j) \overline{a_j}(\tilde{x}_j). Q_j$ and $a_j : L_j$. We say that S is *effective* if, for all $A \neq L_j$,

1. if for some j , C_j , is not A -closed then S persistently manages all $x : A$, and
2. S uniquely handles all names $x : L_j$ for all j .

Thus, a friendly agent is effective if it persistently manages, that is allows no sharing of, all of the names that appear on the interface of its components, except those which provide access to the replicators of the system which are uniquely handled. It is easy to modify the conditions above to obtain results guaranteeing the effectiveness of systems:

Proposition 2.5.11 Let $S = (\nu \tilde{x})(\Pi P_i \mid \Pi! C_j)$ be a friendly agent where $C_j \equiv (\nu \tilde{y}_j) \overline{a_j}(\tilde{x}_j). Q_j$ and $a_j : L_j$. Suppose that for all sorts $A \neq L_j$, such that C_k is not A -closed for some k , the following hold:

1. if $A \in \bigcup \lambda(L)$ for some L then S is L -closed;

2. if $x \in \text{fn}(C_j)$ then $x : L_k$ and, x is not handled in Q_j ;
3. if $a_i = a_j$ then $i = j$;
4. if P is a derivative of C_j , $P \xrightarrow{\alpha} P'$, $\alpha = (\nu \tilde{w})\bar{a}(\dots, y, \dots)$ and $y : A \neq L_j$, then y is not controlled in P' ;
5. if P is a derivative of C_j , $P \xrightarrow{\alpha} P'$ and $\alpha = a(\dots, y, \dots)$, $y : A \neq L_j$, then y is not borne in P' ;
6. if P is a derivative of C_j , $P \xrightarrow{\alpha} P'$ and $\alpha = a(\dots, y, \dots)$, $y : L_j$, then y is not handled in P' .

Then S is effective.

PROOF: It is easy to see by the previous proposition that a friendly agent S , satisfying the conditions of the proposition, persistently manages all names $x : A$, $A \neq L_j$. The proof that all names of sort L_j are uniquely handled in S involves showing that each such name is uniquely handled by one of the replicators of S . This is by induction on the length of the derivation and it makes use of Conditions 2,3 and 6. \square

Finally, we make an observation concerning the use of the notion of controlling as opposed to that of handling. It is not difficult to see that the former notion is essential for the soundness of the results we have presented. For example let $P_1 = \bar{a}(x).P'_1$, $P_2 = \bar{x}(y).P'_2$, $P_3 = a(y).\bar{y}(z).P'_3$ and consider $S \equiv P_1 \mid P_2 \mid P_3$. We may see that exactly one component handles x , namely P_2 . However, S may engage in the transition

$$S \longrightarrow S' = P'_1 \mid \bar{x}(y).P'_2 \mid \bar{x}(z).P'_3$$

and x is handled by two components of S' . This implies that the ability of sending a name may subsequently result in its handling. Thus to ensure unique handling it is necessary to take account of how names are controlled.

We conclude with a summary of the main definitions of this section.

Summary of definitions

Given a friendly agent

$$S = (\nu \tilde{x})(\Pi P_i \mid \Pi! C_j)$$

where $C_j \equiv (\nu \tilde{y}_j)\bar{a}_j(\tilde{x}_j).Q_j$ and $a_j : L_j$, given a name x , we have the following:

- a component P of S bears x , or x is borne in P , if x occurs free in P in a positive subject position;

- a component P of S *handles* x , or x is handled in P , if x occurs free in P in a negative subject position;
- a component P of S *controls* x , or x is controlled in P , if x occurs free in P in negative subject position or in object position;
- S *manages* x , or x is managed in S , if either
 1. $x \in \text{bn}(S)$ and x is borne by at most one component of S and controlled by at most one component of S , or
 2. $x \in \text{fn}(S)$ and either x is borne by at most one component of S , or x is controlled by at most one component of S but not both;
- S *persistently manages* x if for all derivatives S' of S , S' manages x ;
- S is *effective* if, for all $A \neq L_j$,
 1. if for some j , C_j , is not A -closed then S persistently manages all $x : A$, and
 2. S uniquely handles all names $x : L_j$ for all j .

Chapter 3

Determinacy and Confluence

In the setting of term rewriting systems, confluence [CR36] has been a subject of thorough study [Hue80]. The investigation of confluence in process calculi was initiated by Milner in [Mil80, Mil89], where confluence for CCS processes was defined and studied. Although many systems are inherently indeterminate, the importance of determinacy and confluence is closely related with predictability. Indeed it is often the case that in designing a concurrent system the intention is that it will behave in a predictable way, in the sense that if run from the same initial state it will always offer the same choices to its environment, or produce the same observable behaviours. One of the main observations made in [Mil80], where the importance of the notions is further discussed, is that trace equivalence coincides with bisimilarity for confluent systems. A main motivation behind the theory developed was to provide a theoretical framework within which one may build confluent systems from confluent components. It turned out that a reasonable class of useful systems can be constructed in this way as a variety of CCS operators and certain derived constructors, including the confluent composition and the confluent sum, preserve confluence.

Subsequently, the notion of confluence was investigated by other authors. In [San82] confluence was studied within the context of value-passing CCS and it was demonstrated that knowledge of the structure of a process may simplify reasoning about its properties. Further, in [Tof90] this study was extended by considering conditions under which combinations of non-confluent but somehow well-behaved agents, called *semi-confluent agents*, yield confluent systems. The theory was used to show that certain syntactic conditions on programs of a concurrent imperative language guarantee determinacy. More recently, work on determinacy was undertaken in [Nes96], where a static type system was presented for a mobile-process

calculus with the intention that well-typed processes be determinate.

A central property of confluent agents is that they are semantically invariant under silent actions. The relationship of this property, referred to as τ -inertness, to confluence was studied in detail by Groote and Sellink in [GS95] where a weaker notion of confluence was introduced and was shown to coincide with τ -inertness for systems free from divergence. In this chapter we study notions of determinacy and confluence within the π_v -calculus and we present a verification of a protocol which employs the theory.

3.1 Determinacy

In [Mil80, Mil89], Milner introduced and studied a precise notion of *determinacy* of CCS agents. According to the definition, a CCS agent P is *strongly determinate* if, for each derivative Q of P and for all actions α ,

$$\text{whenever } Q \xrightarrow{\alpha} Q' \text{ and } Q \xrightarrow{\alpha} Q'' \text{ then } Q' \sim Q'',$$

where \sim denotes CCS bisimilarity. This notion carries over straightforwardly to the early treatment of the π_v -calculus. It is expressed as follows:

Definition 3.1.1 P is *strongly determinate* if, for every derivative Q of P and for all $\alpha \in Act^+$, whenever $Q \xrightarrow{\alpha} Q'$ and $Q \xrightarrow{\alpha} Q''$ then $Q' \sim Q''$.

This definition makes precise the requirement that when an experiment, a , is conducted on a process P , it should always lead to the same state up to bisimulation. For example, consider the processes $a.(b.0 + c.0)$ and $a.b.0 + a.c.0$. The first process is determinate since after engaging in the action a it always offers the choice of b and c . However, the latter process has two different a -derivatives and therefore it is not determinate.

Note that an additional feature present in this setting is *name instantiation*. Unlike value-passing CCS, passing of values between π_v -calculus processes includes passing of names which can subsequently be used for communication, so it is necessary to ensure that all possible input actions do not cause interference with the names already known to a process and thus preserve determinacy. For example, consider the process

$$B \equiv a(x).(x(y).\bar{a}\langle y \rangle.0 + b(y).0).$$

It can engage in the input action $a\langle b \rangle$ and become B' as follows:

$$B \xrightarrow{a\langle b \rangle} B' \equiv b(y).\bar{a}\langle y \rangle.0 + b(y).0$$

Clearly, B' is not determinate since it has two different $b(y)$ -derivatives. Thus B is indeterminate.

This notion of determinacy is well-behaved, as the following proposition holds.

Proposition 3.1.2 Strong determinacy is closed under derivation; that is, if P is strongly determinate and $P \xrightarrow{\alpha} Q$ then Q is strongly determinate. Strong determinacy is preserved by strong bisimilarity; that is, if P is strongly determinate and $P \sim Q$ then Q is strongly determinate.

PROOF: The proof follows easily by definition. \square

As observed in [Mil89], strong determinacy does not capture the notion of predictability in a very satisfactory way. For example, consider the process

$$A \equiv a.0 + \tau.0.$$

According to the definition, A is strongly determinate. However, its ability to preclude a by doing a τ makes the process unpredictable. Therefore the following variant of strong determinacy concentrating on observable behaviour is introduced.

Definition 3.1.3 P is *weakly determinate* if, for every derivative Q of P and for all $\alpha \in Act^+$, whenever $Q \xrightarrow{\alpha} Q'$ and $Q \xrightarrow{\hat{\alpha}} Q''$ then $Q' \approx Q''$.

For the agent A defined above, we have that $A \Rightarrow A$ and $A \xrightarrow{\tau} 0$ so, since $A \not\approx 0$, A is *not* weakly determinate. The notion of weak determinacy is of greater interest to us so we will henceforth use the term 'determinacy' to refer to weak determinacy. We can now record the fact that (weak) determinacy behaves well.

Proposition 3.1.4 Determinacy is closed under derivation and is preserved by weak bisimilarity. \square

We can give the following characterization of determinacy which considers a sequence of observable actions as opposed to a single action.

Proposition 3.1.5 P is determinate iff for all $s \in Act^*$, whenever $P \xRightarrow{s} P_1$ and $P \xRightarrow{s} P_2$ then $P_1 \approx P_2$.

PROOF: It is easy to see that determinacy implies the property set out in the new characterization. To show the converse, it suffices to note that the new definition is preserved under derivation. \square

A main result presented in [Eng85] (and later in [Mil89] in the context of CCS processes) concerning determinacy is that weak bisimilarity coincides with

trace equivalence for determinate parallel processes. We also record this fact in the π_v -calculus setting, where trace equivalence is defined as follows:

Definition 3.1.6 Agents P and Q are *trace equivalent*, $P \approx_{tr} Q$, if, for all $s \in Act^*$, $bn(s) \cap fn(P, Q) = \emptyset$, $P \xRightarrow{s}$ iff $Q \xRightarrow{s}$.

Trace equivalence is too weak a relation as a criterion for observation equivalence in general, since it does not respect deadlock. However, if we know that processes P and Q are both determinate, then the following shows that in order to establish $P \approx Q$ it is enough to show $P \approx_{tr} Q$.

Theorem 3.1.7 If processes P and Q are determinate then $P \approx Q$ iff $P \approx_{tr} Q$.

PROOF: The implication from left to right holds since $\approx \subseteq \approx_{tr}$. To prove the converse we need to show that

$$\mathcal{B} = \{(P, Q) \mid P \approx_{tr} Q, P, Q \text{ are determinate}\}$$

is a weak bisimulation. The details are not hard and the import of determinacy is clearly exhibited in showing that if $P \xrightarrow{\alpha} P'$ and $Q \xRightarrow{\hat{\alpha}} Q'$ then $P' \approx_{tr} Q'$. \square

3.2 Preserving Determinacy

Since predictable behaviour is a requirement for a variety of systems it appears worthwhile to investigate the possibility of designing compositional methods for constructing determinate systems. It is easy to realise that not all systems built from determinate components are determinate. However, we would like to find some design rules which, without being too restrictive, ensure that by using determinate components we must arrive at a determinate system. We begin by examining which of the π_v -calculus combinators preserve determinacy.

Proposition 3.2.1 If P, P_1, P_2 are determinate, then so are the following:

1. $0, \tau.P, (\nu \tilde{y})\bar{a}\langle \tilde{x} \rangle.P, (\nu x)P.$
2. $a(\tilde{x}).P$, provided that for all i , if $x_i : T_i$ then for all $y \in fn(P)$, if $y : T_i$ then $x_i = y$, i.e. x_i is the only name of its sort that may occur free in P .
3. $\sum_{i \in I} \alpha_i.P_i$, $\alpha_i \neq \tau$, provided that $\alpha_i.P_i$ is determinate for all i and the $\text{subj}(\alpha_i)$ are pairwise distinct.
4. $P_1 \mid P_2$, provided that $fn(P_1) \cap fn(P_2) = \emptyset$ and $\text{sort}(bn(P_1)) \cap \text{sort}(n(P_2)) = \emptyset$, $\text{sort}(bn(P_2)) \cap \text{sort}(n(P_1)) = \emptyset$.

PROOF: The proof of Clauses (1) and (3) is straightforward from the definition. The proof of Clause (2) is more interesting. Note that the side condition requires that each of the names occurring in the object of the input prefix is the *only* name of its sort occurring free in P . We may then deduce that performing the input will not result in the receipt of names that will interfere with the names already known in P . To prove the result it suffices to show that if P is determinate, $x : T$, and $\text{fn}(P)[T = \{x\}]$ then $P\{y/x\}$ is determinate. The proof follows by structural induction on P .

Finally, Clause 4 asserts that the parallel composition of two determinate agents which share no free names and cannot communicate with each other (as achieved by the restrictions on sorts) is determinate. The proof is easy and is omitted. \square

We proceed to examine why we have to impose such strong side-conditions in Clauses (2) and (4). First consider Clause (2) and let P be the following process:

$$P \stackrel{\text{def}}{=} b(x). \bar{x}(z) + c(y). \bar{d}(y)$$

Further, suppose that all names of P have the same sort. It is easy to see that P is a determinate process. However, consider $P\{c/b\}$. This is $c(x). \bar{x}(z) + c(y). \bar{d}(y)$, which is clearly indeterminate. Thus, determinacy is not preserved by name substitution. Consequently, determinacy is not preserved by input prefix since, for example, $a(b).P$ has $P\{c/b\}$ as a derivative. Nonetheless, we may expect that the process $a(\tilde{x}).P$ will be determinate if we can guarantee that the names received in the input are not free in P . In order to enforce this, Clause (2) demands that for all $x \in \tilde{x}$, x is the only name of its sort occurring free within P . Returning to the process P above we may see that if b and c are names of distinct sorts, $\{c/b\}$ is not a valid substitution and $a(b).P$ is a determinate process.

Moving to parallel composition, we first note that it is necessary for P_1 and P_2 to have disjoint sets of free names for reasons similar to those of the CCS case (for example, for $A \stackrel{\text{def}}{=} a.0 \mid \bar{a}.0$, $A \xrightarrow{\tau} 0$, $A \xrightarrow{a} \bar{a}.0$). However, in a mobile calculus setting, this condition is inadequate as there is no guarantee that it will be preserved during computation. For instance, let

$$P \stackrel{\text{def}}{=} w(z). z(x). \bar{b}.0$$

$$Q \stackrel{\text{def}}{=} a(y). \bar{c}.0$$

where a and z are of the same sort, and consider $P \mid Q$. It may engage in the transition

$$P \mid Q \xrightarrow{w(a)} S \equiv a(x). \bar{b}.0 \mid a(y). \bar{c}.0$$

and evolve to S which has two inequivalent $a(w)$ -derivatives. In order to tackle this problem, the side-condition of Clause 4 requires that P_1 and P_2 have no free names in common and, additionally, that the sorts of each agent's bound names are distinct from the sorts of all the names of its companion. In this way, the bound names of each of the agents may not be instantiated to a name already occurring in the other one. Indeed, it is easy to see that if z and a , in the processes P, Q above, are of distinct sorts then S is not a derivative of $P \mid Q$ and this latter agent is determinate.

We conclude this section with some simple results which will be useful in later chapters.

Proposition 3.2.2 Suppose each of the processes P_0, \dots, P_n, Q, R , is determinate and let $d : T, v_1 : T_1, \dots, v_n : T_n$, for some sorts T, T_1, \dots, T_n . Further, suppose that if $Q \xrightarrow{s} \xrightarrow{\bar{d}} Q'$, where $d \notin \text{obj}(s)$, then $Q' \approx 0$ and if $P_i(v) \xrightarrow{s} \xrightarrow{\bar{v}(x)} P'_i$, where $v \notin \text{subj}(s)$, then $P'_i \approx 0$. Then, assuming that they are T, \tilde{T} -closed, the following are determinate:

1. $(\nu d)(Q \mid d.R)$,
2. $S \stackrel{\text{def}}{=} (\nu d)(Q \mid d.S)$,
3. $(\nu \tilde{w} \tilde{v})(P_0(v_0) \mid w_1.P_1(v_1) \mid \dots \mid w_n.P_n(v_n) \mid v_0(x_0). \overline{w_1}.v_1(x_1). \dots \overline{w_n}.v(x_n). \overline{w_0}(\tilde{x}).R)$, where $\tilde{x} \notin \text{fnv}(R)$.

PROOF: The proof consists of examining the derivatives of each of the processes and ensuring that they satisfy the determinacy property. It is easy to see that all such derivatives have exactly one active component (unless they represent a state where a communication via d or a v_i is possible) and by the determinacy of the individual components the result follows. Note that the side condition in 3 is essential as determinacy is not preserved by name substitution. Further, the condition that the agents are T, \tilde{T} -closed guarantees that names d, \tilde{v} , remain private during execution. This is important as extrusion of their scope may result in activation of more than one component and thus indeterminacy, as exhibited by the following process:

$$\begin{aligned} P &\stackrel{\text{def}}{=} (\nu d)(\bar{x}(d).a.\bar{d}.0 \mid d.a.b.0) \xrightarrow{\bar{x}(d)} a.\bar{d}.0 \mid d.a.b.0 \\ &\xrightarrow{d} a.\bar{d}.0 \mid a.b.0 \end{aligned}$$

□

3.3 \asymp -determinacy

In our analysis so far, we have taken weak bisimilarity as the requirement for behavioural equivalence. In this section we consider alternative definitions of deter-

minacy in terms of other equivalences and investigate their relation to the original notion. Letting \asymp range over $\simeq, \approx, =_{wc}, \approx_{tr}$, we define \asymp -determinacy as follows:

Definition 3.3.1 An agent P is \asymp -determinate if, for every derivative Q of P , if $Q \xrightarrow{\alpha} Q_1, Q \xrightarrow{\hat{\alpha}} Q_2$ then $Q_1 \asymp Q_2$.

Thus, we have branching determinacy (for \simeq), (weak) determinacy (for \approx), coupled determinacy (for $=_{wc}$), and trace determinacy (for \approx_{tr}). It is straightforward to see that the following holds:

Proposition 3.3.2 \asymp -determinacy is closed under derivation. \square

Furthermore, we say that an agent Q is τ_{\asymp} -inert if, for each of its derivatives P , whenever $P \Rightarrow P', P \asymp P'$. It is easy to see that a \asymp -determinate agent is τ_{\asymp} -inert.

Proposition 3.3.3 If P is \asymp -determinate then P is τ_{\asymp} -inert.

PROOF: This is a direct consequence of the definition. \square

Although $\simeq \subset \approx_{tr}$ is a strict inclusion, we may see that branching bisimilarity coincides with trace bisimilarity for $\tau_{\approx_{tr}}$ -inert agents. Note that the result extends Theorem 3.1.7.

Proposition 3.3.4 If P, Q are $\tau_{\approx_{tr}}$ -inert then $P \simeq Q$ iff $P \approx_{tr} Q$.

PROOF: Since $\simeq \subset \approx_{tr}$, $P \simeq Q$ implies $P \approx_{tr} Q$ without the assumption of τ -inertness.

To prove the converse, let \mathcal{R} be the following relation

$$\mathcal{R} = \{(P, Q) \mid P \approx_{tr} Q, P, Q \text{ are } \tau_{\approx_{tr}}\text{-inert}\}.$$

We will show that \mathcal{R} is a branching bisimulation. So let $(P, Q) \in \mathcal{R}$ and suppose $P \xrightarrow{\alpha} P'$. If $\alpha = \tau$ then $P \approx_{tr} P'$ by τ -inertness and so $(P', Q) \in \mathcal{R}$ as required. Otherwise, if $\alpha \neq \tau$, then there exist Q', Q'', R such that $Q \Rightarrow Q'' \xrightarrow{\alpha} Q' \Rightarrow R \approx_{tr} P'$. By τ -inertness $Q \approx_{tr} Q''$ and $Q' \approx_{tr} R$. Hence, $(P, Q'') \in \mathcal{R}$ and $(P', Q') \in \mathcal{R}$, which completes the proof. \square

As a corollary we have that τ_{\simeq} -inertness and $\tau_{\approx_{tr}}$ -inertness (and, since $\approx_{tr} \supset =_{wc} \supset \approx \supset \simeq$, all notions of τ_{\asymp} -inertness) coincide.

Proposition 3.3.5 P is τ_{\simeq} -inert iff P is $\tau_{\approx_{tr}}$ -inert.

PROOF: The result in the left to right direction is a direct consequence of the fact that $\simeq \subseteq \approx_{tr}$. To prove the converse suppose P is $\tau_{\approx_{tr}}$ -inert and $P \longrightarrow P'$. Then $P \approx_{tr} P'$ and by the previous proposition $P \simeq P'$. SO, P is τ_{\simeq} -inert as required. \square

Thus we may deduce that all notions of determinacy coincide. We have:

Proposition 3.3.6 P is \approx_{tr} -determinate iff P is \simeq -determinate.

PROOF: By Proposition 3.3.4, the definition of \approx_{tr} -determinacy may be rewritten using \simeq instead of \approx_{tr} , since all agents involved are \approx_{tr} -determinate and thus $\tau_{\approx_{tr}}$ -inert. Thus \approx_{tr} -determinacy implies \simeq -determinacy. Similarly, the characterization of \simeq -determinacy may be rewritten using \simeq instead of \approx_{tr} since all the agents involved are \simeq -determinate and thus $\tau_{\approx_{tr}}$ -inert. Therefore, \simeq -determinacy implies \approx_{tr} -determinacy which completes the proof. \square

Finally we observe that determinacy is preserved by $=_{wc}$ and hence by the other equivalences finer to it. Note however, that determinacy is not preserved by trace equivalence. A counter-example is given by the following processes: $A = a.b.0$ is determinate and $A \approx_{tr} B = a.b.0 + a.0$, but B is not determinate.

Proposition 3.3.7 If P is determinate and $P =_{wc} Q$ then Q is determinate.

PROOF: Let P be a determinate process and suppose $P =_{wc} Q$, where (S_1, S_2) is a weak coupled simulation relating P and Q . Let $Q \xrightarrow{\alpha} Q_1$ and $Q \xrightarrow{\hat{\alpha}} Q_2$. Then there exist P'_1, P'_2 such that $P \xrightarrow{\alpha} P_1$, $P \xrightarrow{\alpha} P_2$ and $P_1 S_2 Q_1$ and $P_2 S_2 Q_2$. By determinacy, $P_1 =_{wc} P_2$. Furthermore, by the definition of $=_{wc}$, $P_1 \Rightarrow P'_1$ and $P_2 \Rightarrow P'_2$ such that $P'_1 S_1 Q_1$ and $P'_2 S_1 Q_2$. By determinacy we also have that $P_1 =_{wc} P'_1$ and $P_2 =_{wc} P'_2$. Thus, $P_1 =_{wc} Q_1$ and $P_2 =_{wc} Q_2$, which implies that $Q_1 =_{wc} Q_2$ as required. \square

3.4 Confluence

The motivation behind the introduction in [Mil89] of confluence for CCS agents was to strengthen determinacy to a notion preserved by a wider set of operators. In this section we extend the definition of confluence to the π_b -calculus and study some of its theory. It turns out that confluence is preserved by a less restrictive form of composition than is determinacy.

According to the definition of [Mil89], a CCS agent P is strongly confluent if it is strongly determinate and for each of its derivatives Q and distinct actions α, β given transitions to Q_1 and Q_2 , the following diagram can be completed.

$$\begin{array}{ccc} Q & \xrightarrow{\alpha} & Q_1 \\ \beta \downarrow & & \beta \downarrow \\ Q_2 & \xrightarrow{\alpha} & Q'_2 \sim Q'_1 \end{array}$$

Let P be the π_v -calculus agent $P \stackrel{\text{def}}{=} a(x).\bar{x}(y).0$ and consider the following transitions.

$$\begin{aligned} P &\xrightarrow{a(b)} \bar{b}(y).0 \\ P &\xrightarrow{a(c)} \bar{c}(y).0 \end{aligned}$$

Clearly, the two transitions cannot be ‘brought together’ in order to complete the diagram above and thus P does not satisfy the definition of confluence just given for CCS agents. However, we would expect that, despite this fact, P should be classified as confluent. Indeed, investigation of confluence in the context of value-passing calculi resulted in extending the CCS definition above to take account of substitution of values, [San82, Tof90]. The definitions highlighted the asymmetry between input and output actions by considering them separately. In the earlier formulation of [San82], the definition for CCS agents was extended to value-passing CCS agents by considering inputs on the same channel as follows:

If $Q \xrightarrow{a(v)} Q_1$ and $Q \xrightarrow{a(u)} Q_2$, then there exist Q' and variable z such that $Q_1 \sim Q'\{v/z\}$ and $Q_2 \sim Q'\{u/z\}$.

An alternative definition subsequently appeared in [Tof90]. This definition makes use of the notion of Π -confluence, where Π is a partition of the set of actions of the transition system in question. A process P is *strongly Π -confluent* if it is strongly determinate and for each derivative Q of P and actions α, β which lie in different blocks of the partition, given transitions to Q_1 and Q_2 , the following diagram can be completed.

$$\begin{array}{ccc} Q & \xrightarrow{\alpha} & Q_1 \\ \beta \downarrow & & \beta \downarrow \\ Q_2 & \xrightarrow{\alpha} & Q'_2 \sim Q'_1 \end{array}$$

It turns out that for value-passing CCS agents strong confluence of [San82] coincides with strong Π -confluence, where Π partitions the set of non-input actions into singletons and collects all input actions of the same channel into one block. So it is not required of a Π -confluent agent that the occurrence of an input action will not preclude that of another performed via the same channel. This type of treatment of input actions carries over nicely to a mobile setting.

A feature distinctive of the π -calculus is the occurrence of bound names in output actions. For let $P \stackrel{\text{def}}{=} (\nu x)\overline{\text{out}}(x).0$. P has the following two transitions (among others).

$$P \xrightarrow{(\nu y)\overline{\text{out}}(y)} 0$$

$$P \xrightarrow{(\nu z)\overline{out}(z)} 0$$

Considering these two transitions to be different would result, counter-intuitively, in characterizing P as a non-confluent process. To avoid this problem, we only require completion of the confluence diagram for actions that are not α -convertible to each other. This is made precise below.

Bound names are the source of another problem, illustrated in the following example. Let $T \stackrel{\text{def}}{=} (\nu z)(\bar{a}(z).0 \mid \bar{b}(z).0)$. It has two transitions.

$$\begin{aligned} T &\xrightarrow{(\nu z)\bar{a}(z)} T_1 \equiv \bar{b}(z).0 \xrightarrow{\bar{b}(z)} 0 \\ T &\xrightarrow{(\nu z)\bar{b}(z)} T_2 \equiv \bar{a}(z).0 \xrightarrow{\bar{a}(z)} 0 \end{aligned}$$

Thus, a computation of T results in creating a new name z and sending it to the environment via names a and b . However, the two possible transitions of each computation are such that z occurs bound only in the first action and consequently free in the second. Hence the confluence diagram cannot be completed in the strict sense, requiring that an occurrence of an action can never preclude the (exact) occurrence of another. Nonetheless, we would like to consider T as confluent. So, instead we require the completion of the confluence diagram presented in the definition below. We proceed straight to weak confluence, which we refer to simply as ‘confluence’. It is useful to distinguish pairs of actions whose components are not α -convertible to each other, and are not inputs via the same name. We employ the following notation.

Notation 3.4.1 For two actions $\beta, \gamma \in Act^+$ we write $\beta \bowtie \gamma$, if $\beta \neq \gamma$, and if $\beta = b\langle \tilde{x} \rangle$ and $\gamma = c\langle \tilde{y} \rangle$, then $b \neq c$.

Moreover, we have the following definition.

Definition 3.4.2 Given two actions α and β we define the *weight* of α over β , written $\alpha \downarrow \beta$, as follows:

$$\alpha \downarrow \beta \stackrel{\text{def}}{=} \begin{cases} (\nu \tilde{u} - \tilde{w})\bar{a}\langle \tilde{x} \rangle, & \text{if } \alpha = (\nu \tilde{u})\bar{a}\langle \tilde{x} \rangle, \tilde{w} = \tilde{u} \cap \text{bn}(\beta) \\ \alpha, & \text{otherwise} \end{cases}$$

Further, given $s, t \in Act^{+*}$, we define the *weight* of s over t , written $s \downarrow t$, inductively upon s and t as below.

$$\begin{aligned} \varepsilon \downarrow t &\stackrel{\text{def}}{=} \varepsilon \\ s \downarrow \varepsilon &\stackrel{\text{def}}{=} s \\ \alpha s \downarrow \beta t &\stackrel{\text{def}}{=} ((\alpha \downarrow \beta) \downarrow t)(s \downarrow \beta t) \end{aligned}$$

Thus, $\alpha|\beta$ is obtained from α by extruding the scope of certain names (the intersection of the bound names of α and β). Similarly, $s|t$ is obtained from s by removing the restriction on all names that occur bound in s and t .

Definition 3.4.3 A process P is *locally confluent* if, for all actions α and β such that $\alpha \bowtie \beta$, the following diagrams can be completed, given transitions from P to P_1 and P_2 .

$$\begin{array}{ccc} P & \xrightarrow{\alpha} & P_1 \\ \alpha \downarrow & & \downarrow \\ P_2 & \Rightarrow & P'_2 \approx P'_1 \end{array} \qquad \begin{array}{ccc} P & \xrightarrow{\alpha} & P_1 \\ \beta \downarrow & & \widehat{\beta|\alpha} \downarrow \\ P_2 & \xRightarrow{\widehat{\alpha|\beta}} & P'_2 \approx P'_1 \end{array}$$

Further, we say that P is *confluent* if, for all derivatives Q of P , Q is locally confluent.

Thus a confluent agent satisfies the confluence diagrams for any two transitions involving actions that are not both inputs via the same name. In addition, if the actions share bound names \tilde{w} , then the transitions used to complete the diagram involve actions where the names \tilde{w} occur free. So, for example, the agent T above is confluent. We have the following property.

Proposition 3.4.4 Weak confluence is closed under derivation and is preserved by weak bisimilarity. \square

We may also easily see that an agent is confluent if the diagrams may be completed ‘up to \approx ’ as below.

Proposition 3.4.5 A process P is confluent iff for all derivatives Q of P , and for all α and $\beta \in \text{Act}^+$ such that $\alpha \bowtie \beta$, given transitions from Q to Q_1 and Q_2 , the following diagrams can be completed.

$$\begin{array}{ccc} Q & \xrightarrow{\alpha} & Q_1 \approx Q'_1 \\ \alpha \downarrow & & \downarrow \\ Q_2 \approx Q''_2 & \Rightarrow & Q'_2 \approx Q'_1 \end{array} \qquad \begin{array}{ccc} Q & \xrightarrow{\alpha} & Q_1 \approx Q'_1 \\ \beta \downarrow & & \widehat{\beta|\alpha} \downarrow \\ Q_2 \approx Q''_2 & \xRightarrow{\widehat{\alpha|\beta}} & Q'_2 \approx Q'_1 \end{array}$$

PROOF: This is straightforward using properties of \approx . \square

Recall that an agent P is τ_{\approx} -inert if, for each of its derivatives Q , whenever $Q \Rightarrow Q'$ then $Q \approx Q'$. Hereafter, we will simply write τ -inert for τ_{\approx} -inert agents. We observe that confluence implies τ -inertness.

Proposition 3.4.6 If P is confluent then P is τ -inert.

PROOF: The proof consists in showing that

$$\mathcal{R} = \{(P, Q) \mid P \Rightarrow \approx Q, P \text{ is confluent}\}$$

is a weak bisimulation. Then, since confluence is closed under derivation, we may conclude that if an agent P is confluent then it is also τ -inert.

So let $(P, Q) \in \mathcal{R}$ and suppose $P \xrightarrow{\alpha} P'$. Then, by confluence and Proposition 3.4.5, $Q \xRightarrow{\hat{\alpha}} Q'$ and $P' \Rightarrow P'' \approx Q'$. Hence $(P', Q') \in \mathcal{R}$ as required. \square

An easy corollary of this result is that confluence implies determinacy.

Proposition 3.4.7 If P is confluent then P is determinate.

PROOF: Straightforward from the previous proposition and the definitions. \square

These observations allow the following simpler characterization of confluence:

Proposition 3.4.8 A process P is confluent iff it is τ -inert and, for all derivatives Q of P ,

1. if $\alpha \in \text{Act}$, $Q \xrightarrow{\alpha} Q_1$, $Q \Rightarrow \xrightarrow{\alpha} Q_2$ then $Q_1 \approx Q_2$, and
2. if $\alpha, \beta \in \text{Act}$, $\alpha \bowtie \beta$, given transitions from Q to Q_1 and Q_2 , the following diagram can be completed.

$$\begin{array}{ccc} Q & \xrightarrow{\alpha} & Q_1 \\ \Downarrow \beta & & \Downarrow \beta \mid \alpha \\ Q_2 & \Rightarrow \xrightarrow{\alpha \mid \beta} & Q'_2 \approx Q'_1 \end{array}$$

PROOF: First, it is easy to see by definition that if P is confluent then the conditions above are satisfied. The converse follows easily by τ -inertness and Proposition 3.4.5. \square

The definitions we have given so far are such that the top line of each diagram is a single action. This formulation is convenient for the purpose of establishing the confluence of agents. However, it would be of great use if we could obtain an alternative characterization which involves single transitions to Q_2 as well as Q_1 . It turns out that this is only possible for the special class of agents described in the following definition. A similar observation regarding τ -inertness was made in [GS95].

Proposition 3.4.9 Let P be a fully convergent agent. Then P is confluent iff it is τ -inert and, for all derivatives Q of P ,

1. if $\alpha \in Act$, $Q \xrightarrow{\alpha} Q_1$, $Q \xrightarrow{\alpha} Q_2$ then $Q_1 \approx Q_2$, and
2. if $\alpha, \beta \in Act$, $\alpha \bowtie \beta$, given transitions to Q_1 and Q_2 , the following diagram can be completed.

$$\begin{array}{ccc} Q & \xrightarrow{\alpha} & Q_1 \\ \beta \downarrow & & \beta|\alpha \Downarrow \\ Q_2 & \xRightarrow{\alpha|\beta} & Q'_2 \approx Q'_1 \end{array}$$

PROOF: It is easy to see that if P is confluent then the diagrams can be completed and P is τ -inert by Proposition 3.4.6. Note that this result follows without the assumption that P is convergent.

To prove the converse, suppose P is fully convergent, τ -inert and the clauses of the proposition are satisfied. We will show that P satisfies the characterization of confluence given in the previous proposition. So, let Q be a derivative of P and suppose

$$Q \xrightarrow{\alpha} Q_1 \text{ and } Q \Rightarrow Q'_2 \xrightarrow{\beta} Q_2$$

where $\alpha, \beta \in Act$, $\alpha \bowtie \beta$. By τ -inertness, $Q \approx Q'_2$. Thus

$$Q'_2 \Rightarrow Q'_1 \xrightarrow{\alpha} R_1 \approx Q_1,$$

where, since Q'_2 is τ -inert, $Q'_2 \approx Q'_1$. Since P is fully convergent, there exists R_2 such that

$$Q'_1 \Rightarrow R_2 \not\rightarrow$$

By τ -inertness $Q'_1 \approx R_2$. Hence $R_2 \xrightarrow{\alpha} Q_3 \approx Q_1$ and $R_2 \xrightarrow{\beta} Q_4 \approx Q_2$, since $R_2 \approx Q'_2$. By the assumption of the proposition,

$$Q_3 \xRightarrow{\beta|\alpha} Q'_3 \text{ and } Q_4 \xRightarrow{\alpha|\beta} Q'_4 \approx Q'_3.$$

Hence

$$Q_1 \Rightarrow \xRightarrow{\beta|\alpha} Q'_1 \approx Q'_3 \text{ and } Q_2 \Rightarrow \xRightarrow{\alpha|\beta} Q'_2 \approx Q'_3$$

and so $Q'_1 \approx Q'_2$ as required.

The case $\alpha = \beta$ follows similarly. □

Note that the assumption of P being fully convergent is essential for the soundness of the proposition (in the right to left direction). A counter-example when the assumption is omitted is given by the process

$$P \stackrel{\text{def}}{=} a.b.0 + \tau.(a.c.0 + \tau.P).$$

It is easy to see that P is τ -inert and that all of its derivatives satisfy the conditions of the proposition. However, $P \xrightarrow{a} P_1 = b.c.0$, $P \xrightarrow{\tau} \xrightarrow{a} P_2 = c.b.0$ and it is not possible to complete the confluence diagram for P_1 and P_2 .

We continue with some results which enunciate some simple but useful properties of confluent agents.

Lemma 3.4.10 Suppose P is a confluent agent and $P \xrightarrow{\gamma} \approx 0$ where $\gamma = (\nu \tilde{u})\bar{a}(\tilde{x})$. Then $P \approx (\nu \tilde{u})\bar{a}(\tilde{x}).0$.

PROOF: Let P be a confluent agent such that $P \Rightarrow Q \xrightarrow{\gamma} R \Rightarrow 0$. Then by τ -inertness $P \approx Q$ and $R \approx 0$. So suppose $Q \xrightarrow{\beta} Q'$. Then the following possibilities exist:

- $\gamma = \beta$ and $Q' \approx 0$ as Q is determinate;
- $\beta \bowtie \gamma$. Then $R \xRightarrow{\beta \upharpoonright \gamma} R'$, $Q' \xRightarrow{\gamma \upharpoonright \beta} Q'' \approx R'$. Since $R \approx 0$, $\beta = \tau$, $R' \approx 0$ and $Q \approx Q'$.

It is thus easy to see that setting

$$\mathcal{B} = \{(P, Q) \mid P \xrightarrow{(\nu \tilde{u})\bar{a}(\tilde{x})} \approx 0, Q \equiv (\nu \tilde{u})\bar{a}(\tilde{x}).0, \text{ and } P \text{ is confluent}\},$$

$\mathcal{B} \cup \approx$ is a weak bisimulation. □

Lemma 3.4.11 Let P be a confluent agent and suppose $P \xrightarrow{\alpha} P_1$ where $\alpha = \bar{x}(\tilde{y})$ and $x \notin \text{fn}(P_1)$. Further, let $s \in \text{Act}^*$ be such that $x \notin \text{obj}(s)$. Then, if $P \xRightarrow{s} P_2$, either $\alpha \in s$, or $x \notin \text{fn}(s)$ and the following diagram can be completed.

$$\begin{array}{ccc} P & \xrightarrow{\alpha} & P_1 \\ s \downarrow & & s \downarrow \\ P_2 & \xRightarrow{\alpha} & P'_2 \approx P'_1 \end{array}$$

PROOF: By induction on s . If s is the empty sequence then $\alpha \notin s$ and by τ -inertness $P \approx P_2$. Hence $P_2 \xRightarrow{\alpha} P'_2 \approx P'_1$ as required.

Otherwise, suppose $s = s_0\beta$ and $P \xRightarrow{s_0} P' \xRightarrow{\beta} P_2$. If $\alpha \in s$ then we are done. So suppose $\alpha \notin s$. Then, by the induction hypothesis, $x \notin \text{fn}(s_0)$ and there exist P', P'', P'_1 such that the following can be completed.

$$\begin{array}{ccc} P & \xrightarrow{\alpha} & P_1 \\ s_0 \downarrow & & s_0 \downarrow \\ P' & \xRightarrow{\alpha} & P'' \approx P'_1 \end{array}$$

Since $P' \xRightarrow{\beta} P_2$ and P' is confluent, $P_2 \xRightarrow{\alpha} P'_2$ and $P'_1 \xRightarrow{\beta} P'_1 \approx P'_2$. Moreover, since $x \notin \text{fn}(P_1)$, by our assumption on s , it is easy to see that $x \notin \text{fn}(P'_1)$ and since $P'_1 \xRightarrow{\beta}$ and $x \notin \text{obj}(\beta)$, it follows that $x \notin \text{fn}(\beta)$ as required. □

Definition 3.4.12 Let $s \in \text{Act}^*$, $\alpha \in \text{Act}^+$. We define the *excess* of s over α , s/α as follows:

$$\begin{aligned} s/\alpha &= s, \text{ if } \alpha = \tau \\ \varepsilon/\alpha &= \varepsilon, \\ (\rho s_0)/\alpha &= s_0, \text{ if } \rho = \alpha \\ &= \rho(s_0/\alpha), \text{ otherwise} \end{aligned}$$

Lemma 3.4.13 Suppose P is confluent, $s \in \text{Act}^*$, $P \xRightarrow{s} P_1$, $P \xRightarrow{\alpha} P_2$ where $\alpha \neq \tau$ and if $\alpha = a\langle \tilde{x} \rangle$ and $s = s_0 \beta s_1$ with $\text{subj}(\beta) = a$, $a \notin \text{subj}(s_0)$, then $\alpha = \beta$. Then either $\alpha \in s$ and $P_2 \xRightarrow{s/\alpha} P_2' \approx P_1$, or $\alpha \notin s$ and the following diagram can be completed.

$$\begin{array}{ccc} P & \xRightarrow{s} & P_1 \\ \alpha \Downarrow & & \alpha \Downarrow \\ P_2 & \xRightarrow{s} & P_2' \approx P_1' \end{array}$$

PROOF: By induction on s . If s is the empty sequence then $\alpha \notin s$ and by τ -inertness $P \approx P_1$. Hence by the definition of \approx the diagram above can be completed.

So suppose $s = s_0 \beta$ and $P \xRightarrow{s_0} P' \xRightarrow{\beta} P_1$. First, if $\alpha \notin s$ then $\alpha \notin s_0$ and by the induction hypothesis $P' \xRightarrow{\alpha} P''$, $P_2 \xRightarrow{s_0} P_2'' \approx P''$. Then as $\alpha \neq \beta$ by confluence and the assumption above, $\text{subj}(\alpha) \neq \text{subj}(\beta)$ and since P' is confluent $P_2'' \xRightarrow{\beta} P_2'$, $P_1 \xRightarrow{\alpha} P_1' \approx P_2'$. Otherwise suppose $\alpha \in s$. If $\alpha \in s_0$ then $P_2 \xRightarrow{s_0/\alpha} P_2'' \approx P'$ and so $P_2 \xRightarrow{s/\alpha} P_1'$. Otherwise, if $\alpha \notin s_0$ then $\alpha = \beta$ and so $P_2'' \approx P_1$ and $P_2 \xRightarrow{s/\alpha} P_2' \approx P_1$ as required. \square

3.5 Preserving Confluence

In this section we consider design rules that allow us to build confluent systems from confluent components. We begin by investigating which of the π_v -calculus operators preserve confluence.

Proposition 3.5.1 If P is confluent then so are the following:

- $\tau. P, (\nu \tilde{z}) \bar{x}(\tilde{y}). P, (\nu x) P.$
- $a(\tilde{x}). P$, provided that if $x_i : T_i$ then for all $y \in \text{fn}(P)$, $y : T_i$, $x_i = y$, i.e. x_i is the only name of its sort occurring free in P for all i .
- $P_1 \mid P_2$, provided that $\text{fn}(P_1) \cap \text{fn}(P_2) = \emptyset$ and $\text{sort}(\text{bn}(P_1)) \cap \text{sort}(\text{n}(P_2)) = \emptyset$, $\text{sort}(\text{bn}(P_2)) \cap \text{sort}(\text{n}(P_1)) = \emptyset$.

PROOF: The proof is routine. \square

Note that, by comparison to Proposition 3.2.1, summation is missing here (for obvious reasons: $a.0$ and $b.0$ are confluent but $a.0 + b.0$ is not).

As mentioned earlier, it is expected that confluence will be preserved by a restricted parallel composition which, unlike the composition of Propositions 3.5.1 and 3.2.1, allows communication between its components. So consider a composition of confluent components. It is easy to see that indeterminacy may arise in the system if two of the components share the ability to interact with a third. The subsequent behaviour of the third component will, in general, not be independent of which interaction takes place. For example, let $P_1 = \bar{a}\langle 3 \rangle$, $P_2 = \bar{a}\langle 5 \rangle$, $P_3 = a(x). \bar{b}\langle x \rangle$. Clearly, the possible computations of $S = P_1 \mid P_2 \mid P_3$ include the following two.

$$\begin{aligned} S &\xrightarrow{\tau} \xrightarrow{\bar{b}\langle 3 \rangle} S_1 = \bar{a}\langle 5 \rangle.0 \\ S &\xrightarrow{\tau} \xrightarrow{\bar{b}\langle 5 \rangle} S_2 = \bar{a}\langle 3 \rangle.0 \end{aligned}$$

Clearly the matching actions required to complete the confluence diagram are not possible. We might expect, however, that if in no state reachable from an agent the capability of using a name (either for input or for output) is shared by two components of the agent, then the agent is confluent. Note that an additional requirement to ensure confluence is that if a name is both borne and handled within a process then it does not occur free. Otherwise the ability of using it is shared between the components and the environment (for example, think of $P_1 \mid P_3$ from above). Hence the notion of persistent management of names within friendly agents (of Chapter 2) plays a crucial rôle in our search of design rules that preserve confluence.

The main result makes use of the following two lemmas. The first asserts that a communication between two confluent components of a friendly agent via a name that is persistently managed does not affect the state of a system up to bisimilarity. The second establishes an analogous result for a communication involving a replicator.

Lemma 3.5.2 Suppose $S \stackrel{\text{def}}{=} (\nu \tilde{z})(\Pi P_i \mid \Pi! C_j)$ is a friendly agent where components P_1 and P_2 are confluent. Suppose $P_1 \xrightarrow{\alpha} P'_1$, $P_2 \xrightarrow{\bar{\alpha}} P'_2$ where $\alpha = a\langle \tilde{x} \rangle$, $\bar{\alpha} = (\nu \tilde{u})\bar{a}\langle \tilde{x} \rangle$. Let $S' = (\nu \tilde{z}\tilde{u})(P'_1 \mid P'_2 \mid \Pi_{i \neq 1,2} P_i \mid \Pi! C_j)$. Then assuming that $a : A$ is persistently managed in S , $S \approx S'$.

PROOF: Let $(S, S') \in \mathcal{B}$ if $S \stackrel{\text{def}}{=} (\nu \tilde{z})(\Pi P_i \mid \Pi! C_j)$ and $S' = (\nu \tilde{z}\tilde{u})(P'_1 \mid P'_2 \mid \Pi_{i \neq 1,2} P_i \mid \Pi! C_j)$, where P_1 and P_2 are confluent, $P_1 \xrightarrow{\alpha} \approx P'_1$, $P_2 \xrightarrow{\bar{\alpha}} \approx P'_2$, $\alpha = a\langle \tilde{x} \rangle$, $\bar{\alpha} = (\nu \tilde{u})\bar{a}\langle \tilde{x} \rangle$, and a is managed in S . We will show that $\mathcal{B} \cup \approx$ is a weak bisimulation. So suppose $(S, S') \in \mathcal{B}$ where S and S' are as above.

Suppose $S' \xrightarrow{\beta} T'$. Clearly, $S \Rightarrow T'' \approx S'$ and so it is straightforward to find T such that $T'' \xrightarrow{\beta} T \approx T'$.

Suppose $S \xrightarrow{\beta} T$. Note that since a is managed in S , and it occurs in both P_1 and P_2 then it must occur bound in S and so $a \neq \text{subj}(\beta)$. Various possibilities exist.

- $T = (\nu \tilde{z} - \tilde{v})(Q_1 \mid \Pi_{i \neq 1} P_i \mid \Pi! C_j)$ where $\text{bn}(\beta) = \tilde{v}$ and $P_1 \xrightarrow{\beta} Q_1$. Then by confluence, $P'_1 \xrightarrow{\widehat{\beta|\alpha}} Q'_1$ and $Q_1 \xrightarrow{\alpha|\beta} Q'_1 \approx Q'_1$. So $S' \xrightarrow{\widehat{\beta}} T' = (\nu \tilde{z} \tilde{u} - \tilde{v})(Q'_1 \mid \Pi_{i \neq 1} P_i \mid \Pi! C_j)$ and $(T, T') \in \mathcal{B}$.
- If P_2 acts alone the arguments are the same as those of the previous case.
- $T = (\nu \tilde{z} - \tilde{v})(P_1 \mid P_2 \mid R)$ where $\text{bn}(\beta) = \tilde{v}$ and $(\Pi_{i \neq 1,2} P_i \mid \Pi! C_j) \xrightarrow{\beta} R$. Then clearly, $S' \xrightarrow{\beta} T' = (\nu \tilde{z} \tilde{u} - \tilde{v})(P'_1 \mid P'_2 \mid R)$ and $(T, T') \in \mathcal{B}$.
- If $\beta = \tau$ where one of P_1 and P_2 interacts with $(\Pi_{i \neq 1,2} P_i \mid \Pi! C_j)$ then the argument is a combination of those above. Note that since a is managed, $a \notin \text{fn}(R)$ and thus the interaction is not via a .
- If $\beta = \tau$, where P_1 and P_2 interact via a name other than a , then again the case follows using arguments of the previous cases.
- Finally, suppose $\beta = \tau$ and $T' = (\nu \tilde{z} - \tilde{u})(Q_1 \mid Q_2 \mid \Pi_{i \neq 1,2} P_i \mid \Pi! C_j)$, where $P_1 \xrightarrow{\sigma} Q_1$, $P_2 \xrightarrow{\bar{\sigma}} Q_2$ and $\text{subj}(\sigma) = a$. Since α is persistently managed in S , P_1 uniquely bears a and P_2 uniquely handles a . Thus $\sigma \in A^+$, $\bar{\sigma} \in A^-$. Recall that by the definition of a friendly agent, P_2 is consistent. Thus $\bar{\alpha} = \bar{\sigma}$ and as a result $\alpha = \sigma$. So, by confluence, $Q_1 \approx P'_1$ and $Q_2 \approx P'_2$. Hence $T \approx S'$. \square

Note that in fact persistent management of names is not necessary for the soundness of this lemma (and the subsequent results). It would be sufficient to ensure that, in the notation of the lemma, for all derivatives Q of S , a is owned by at most two components.

Lemma 3.5.3 Suppose $S \stackrel{\text{def}}{=} (\nu \tilde{z})(\Pi_{1 \leq i \leq n} P_i \mid \Pi! C_j)$ is a friendly agent where component P_1 is confluent. Suppose $P_1 \xrightarrow{\alpha} \approx P'_1$, $!C_k \xrightarrow{\bar{\alpha}} P_{n+1} \mid !C_k$ where $\alpha \text{ comp } \bar{\alpha}$ and $\tilde{u} = \text{bn}(\alpha) \cup \text{bn}(\bar{\alpha})$. Let $S' = (\nu \tilde{z} \tilde{u})(P'_1 \mid P_{n+1} \mid \Pi_{i \neq 1} P_i \mid \Pi! C_j)$. Then, assuming that $\text{subj}(\alpha)$ is uniquely handled in S , $S \approx S'$.

PROOF: The proof is similar to that of the previous lemma, only easier, since C_k is not able to act on its own and there exists exactly one communication between P_1 and C_k , namely the one performed in the transition $S \xrightarrow{\tau} S'$. \square

The main result follows.

Theorem 3.5.4 Suppose $S_0 \equiv (\nu\tilde{p})(\Pi P_i \mid \Pi!(\nu\tilde{y})\alpha_j.Q_j)$ is an effective friendly agent, where P_i and Q_j are confluent for all i, j . Then S_0 is confluent.

PROOF: Let $S \equiv (\nu\tilde{p})(\Pi_{1 \leq k \leq m} P_k \mid \Pi! C_j)$ be a derivative of S_0 . It suffices by Proposition 3.4.8 to prove that for all $\alpha, \beta \in \text{Act}$, $\alpha \bowtie \beta$, the following hold.

1. If $S \xrightarrow{\tau} S'$ then $S \approx S'$.
2. If $S \xrightarrow{\alpha} S_1$ and $S \Rightarrow \xrightarrow{\alpha} S_2$, then $S_1 \approx S_2$,
3. If $S \xrightarrow{\alpha} S_1$ and $S \Rightarrow \xrightarrow{\beta} S_2$, then $S_1 \Rightarrow \xrightarrow{\beta \mid \alpha} S'$ and $S_2 \Rightarrow \xrightarrow{\alpha \mid \beta} \approx S'$,

Proof of 1: The following possibilities exist for $S \xrightarrow{\tau} S'$:

- $S' = (\nu\tilde{p})(P'_1 \mid P_2 \mid \dots \mid P_m \mid \Pi! C_j)$, $P_1 \xrightarrow{\tau} P'_1$. By definition of a friendly agent, P_1 is a derivative of some C_j so, since C_j is confluent and confluence is closed under derivation, P_1 is also confluent. Thus, by τ -inertness, $P_1 \approx P'_1$ and $S \approx S'$ as required.
- $S' = (\nu\tilde{p}\tilde{u})(P'_1 \mid P'_2 \mid P_3 \mid \dots \mid P_m \mid \Pi! C_j)$, $P_1 \xrightarrow{\alpha} P'_1$ and $P_2 \xrightarrow{\bar{\alpha}} P'_2$, where $\alpha \text{ comp } \bar{\alpha}$, $\tilde{u} = \text{bn}(\alpha) \cup \text{bn}(\bar{\alpha})$. Then by Lemma 3.5.2, $S \approx S'$.
- $S' = (\nu\tilde{p}\tilde{u})(P'_1 \mid P_2 \mid \dots \mid P_m \mid P_{m+1} \mid \Pi! C_j)$, $P_1 \xrightarrow{\alpha} P'_1$, $!C_i \xrightarrow{\bar{\alpha}} !C_i \mid P_{m+1}$, where $\alpha \text{ comp } \bar{\alpha}$, $\tilde{u} = \text{bn}(\alpha) \cup \text{bn}(\bar{\alpha})$. By Lemma 3.5.3, we also have that $S \approx S'$.

This completes the proof of 1.

Proof of 2: Suppose

$$S \xrightarrow{\alpha} S_1 = (\nu\tilde{p} - \tilde{u})(Q \mid P_2 \mid \dots \mid P_m \mid \Pi! C_j)$$

where $P_1 \xrightarrow{\alpha} Q$, $\tilde{u} = \text{bn}(\alpha)$, and

$$\begin{aligned} S &\Rightarrow S'_2 = (\nu\tilde{p}\tilde{v})(\Pi_{1 \leq k \leq m} P'_k \mid P_{m+1} \mid \dots \mid P_n \mid \Pi! C_j) \\ &\xrightarrow{\alpha} S_2 = (\nu\tilde{p}\tilde{v} - \tilde{u})(P'_1 \mid \dots \mid P'_i \mid \dots \mid P_n \mid \Pi! C_j) \end{aligned}$$

where $\Pi! C_j \xRightarrow{s} P_{m+1} \mid \dots \mid P_n \mid \Pi! C_j$, for each k , $P_k \xRightarrow{s_k} P'_k$ for some s_k , and $P'_i \xrightarrow{\alpha} P''_i$. Since $S \Rightarrow S'_2$ is an internal communication each $x \in \text{subj}(s_k)$ must be both borne and handled in some S' where $S \Rightarrow S' \Rightarrow S'_2$. Thus by the persistent management property, x occurs bound in S' and consequently, x does not occur free in S . Note that since $\text{subj}(\alpha)$ occurs free in S , $\alpha \notin s_1$. Now since $P_1 \xrightarrow{\alpha} Q$, $P_1 \xRightarrow{s_1} P'_1$ and P_1 is confluent, by Lemma 3.4.13, there exists Q'' , Q' such that $Q \xRightarrow{s_1 \mid \alpha} Q'$ and $P'_1 \xRightarrow{\alpha \mid s_1} Q'' \approx Q'$. Since $\text{subj}(\alpha)$ is persistently managed in S_2 and P'_1

owns $\text{subj}(\alpha)$, $i = 1$ and $P'_1 \xrightarrow{\alpha} P''_1$, $P'_1 \xRightarrow{\alpha} Q''$. By confluence $P''_1 \approx Q''$ and so $P'_1 \approx Q'$. Since $Q \xRightarrow{\beta} Q'$,

$$S_1 \Rightarrow S'_1 = (\nu \tilde{p}\tilde{v} - \tilde{u})(Q' \mid \Pi_{2 \leq k \leq m} P'_k \mid P_{m+1} \mid \dots \mid P_n \mid \Pi! C_j)$$

and since $Q' \approx P''_1$, $S'_1 \approx S_2$. This completes the proof of 2.

Proof of 3: Suppose

$$S \xrightarrow{\alpha} S_1 = (\nu \tilde{p} - \tilde{u})(Q \mid \Pi_{k \neq 1} P_k \mid \Pi! C_j)$$

where $P_1 \xrightarrow{\alpha} Q$, $\tilde{u} = \text{bn}(\alpha)$, and for some $\beta \in \text{Act}$, $\alpha \bowtie \beta$,

$$\begin{aligned} S &\Rightarrow S''_2 = (\nu \tilde{p}\tilde{v})(\Pi_{1 \leq k \leq m} P'_k \mid P_{m+1} \mid \dots \mid P_n \mid \Pi! C_j) \\ &\xrightarrow{\beta} S_2 = (\nu \tilde{p}\tilde{v} - \tilde{w})(P'_1 \mid \dots \mid T \mid \dots \mid P_n \mid \Pi! C_j) \end{aligned}$$

where $\tilde{w} = \text{bn}(\beta)$, $\Pi! C_j \xRightarrow{s} P_{m+1} \mid \dots \mid P_n \mid \Pi! C_j$, $P_k \xRightarrow{s_k} P'_k$ for some s_k for all k , and $P'_i \xrightarrow{\beta} T$.

Using arguments similar to those of the previous case we can see that

$$S_1 \Rightarrow S'_1 = (\nu \tilde{u}')(Q' \mid \Pi_{k \neq 1} P'_k \mid P_{m+1} \mid \dots \mid P_n \mid \Pi! C_j)$$

such that $P'_1 \xRightarrow{\alpha} P''_1 \approx Q'$. Two cases exist depending on whether or not $i = 1$. If $i = 1$, since $P'_1 \xRightarrow{\alpha} P''_1 \approx Q'$, $P'_1 \xrightarrow{\beta} T$ and P'_1 is confluent, then $Q' \xRightarrow{\beta|\alpha} Q''$, $T \xRightarrow{\alpha|\beta} T' \approx Q''$. So,

$$S''_1 \xRightarrow{\beta|\alpha} S'_1 = (\nu \tilde{p}\tilde{v} - \tilde{u}\tilde{w})(Q'' \mid \Pi_{k \neq 1} P'_k \mid P_{m+1} \mid \dots \mid P_n \mid \Pi! C_j)$$

and

$$S_2 \xRightarrow{\alpha|\beta} S'_2 = (\nu \tilde{p}\tilde{v} - \tilde{u}\tilde{w})(T' \mid \Pi_{k \neq 1} P'_k \mid P_{m+1} \mid \dots \mid P_n \mid \Pi! C_j).$$

Since $T' \approx Q''$, $S'_1 \approx S'_2$ as required. Otherwise, if $i \neq 1$,

$$S''_1 \xRightarrow{\beta|\alpha} S'_1 = (\nu \tilde{p}\tilde{v} - \tilde{u}\tilde{w})(Q' \mid T \mid \Pi_{k \neq 1, i} P'_k \mid P_{m+1} \mid \dots \mid P_n \mid \Pi! C_j)$$

and

$$S''_2 \xRightarrow{\alpha|\beta} S'_2 = (\nu \tilde{p}\tilde{v} - \tilde{u}\tilde{w})(P''_1 \mid T \mid \Pi_{k \neq 1, i} P'_k \mid P_{m+1} \mid \dots \mid P_n \mid \Pi! C_j)$$

and since $P''_k \approx Q'$, $S'_1 \approx S'_2$ as required.

This completes the proof. \square

Note that if the term ‘confluent’ were replaced by the term ‘determinate’, Theorem 3.5.4 would not hold. For example, consider the agent $S = (\nu a)(\bar{a} + \bar{b} \mid a)$.

Although the processes considered in the parallel composition are determinate and no names are shared we have that $S \approx \bar{b} + \tau$ which is clearly indeterminate.

It is also often the case that while a process is not fully confluent, when placed in a certain context its non-confluent behaviour is not uncovered. In particular, consider a process P such that for each of its derivatives P' there exist a set \tilde{p} of pairs of names such that P' may complete the confluent diagrams for all actions α, β with $(\text{subj}(\alpha), \text{subj}(\beta))$ not in \tilde{p} . Suppose P is placed in a context such that when P becomes P' , the context blocks communications via at least one of a, b for all pairs (a, b) in \tilde{p} . Then we might expect that, assuming the context is somehow well-behaved, the resulting composition will be confluent.

The following definition captures the structure of a class of agents of the kind just considered.

Definition 3.5.5 Suppose L, M, N are sorts and λ a sorting such that $\lambda(L) = \{(M, N)\}$, and for all sorts S if $M, N \in \bigcup \lambda(S)$ then $S = L$. A derivation-closed set \mathcal{S} of processes is L_{MN} -sensitive if there is a partition, an L_{MN} -sensitive partition, $\{\mathcal{S}^{\tilde{p}} \mid \tilde{p} \text{ a finite subset of } M \times N\}$ of \mathcal{S} such that:

1. if $P \in \mathcal{S}^{\tilde{p}}$ and $P \xrightarrow{\alpha} P'$, where $\alpha \notin L^{\pm} \cup M^+ \cup N^+$, then $P' \in \mathcal{S}^{\tilde{p}}$;
2. if $P \in \mathcal{S}^{\tilde{p}}$ and $P \xrightarrow{\alpha} P'$, $\alpha \in L^-$, then $\alpha = (\nu m, n)\bar{s}(m, n)$ for some $s : L$ and $P' \in \mathcal{S}^{\tilde{p}(m, n)}$;
3. if $P \in \mathcal{S}^{\tilde{p}}$ and $P \xrightarrow{s(m, n)} P'$, where $s : L$, then at most one of m and n occurs free in P' ;
4. if $P \in \mathcal{S}^{\tilde{p}}$ and $P \xrightarrow{\alpha} P'$, where $\alpha \in M^+ \cup N^+$, then there exists $(m, n) \in \tilde{p}$ such that $\text{subj}(\alpha) = m$ or $\text{subj}(\alpha) = n$ and $P' \in \mathcal{S}^{\tilde{p}-(m, n)}$.

Further, \mathcal{S} is L_{MN} -confluent, with L_{MN} -confluent partition, $\{\mathcal{S}^{\tilde{p}}\}_{\tilde{p}}$, if it is L_{MN} -sensitive and whenever $P \in \mathcal{S}^{\tilde{p}}$ and $P \xrightarrow{\alpha} P_1$, $P \xrightarrow{\beta} P_2$ then, unless $\alpha \in M^+$, $\beta \in N^+$ with $(\text{subj}(\alpha), \text{subj}(\beta)) \in \tilde{p}$, $P_1 \xrightarrow{\widehat{\beta|\alpha}} P'_1$ and $P_2 \xrightarrow{\widehat{\alpha|\beta}} P'_2 \approx P'_1$. \square

Hence an L_{MN} -confluent set of agents \mathcal{S} is partitioned into sets $\mathcal{S}^{\tilde{p}}$ so that if $P \in \mathcal{S}^{\tilde{p}}$ then P is locally confluent for all pairs of actions other than those whose subjects are a pair in \tilde{p} . Note that the index \tilde{p} of a block of the partition is augmented by actions of the form $(\nu m, n)\bar{s}(m, n)$ where $s : L$. Moreover, by Property 3, whenever an agent of \mathcal{S} receives a pair of names (m, n) via a name of sort L , it discards one of m and n . Thus, it is expected that if we compose in parallel agents of \mathcal{S} , although the individual components are not confluent, their non-confluent behaviour will be lost. This is substantiated below.

Lemma 3.5.6 Let L, M, N be sorts and \mathcal{S} be L_{MN} -confluent with L_{MN} -confluent partition $\{\mathcal{S}^{\tilde{p}}\}_{\tilde{p}}$. If $Q_0 = (\nu \tilde{p})(\Pi_{i \in I} P_i)$ is a fully convergent agent where

1. $P_i \in \mathcal{S}^{\emptyset}$ and P_i is consistent for all i ;
2. for all derivatives $Q \equiv (\nu \tilde{u})(\Pi_{i \in I} P'_i)$ of Q_0 , where P'_i is a derivative of P_i ,
 - (a) if $x : L$ and $x \in \text{fn}(Q)$, then x is not handled in Q ;
 - (b) if $x \in \text{bn}(Q)$ then x is owned by at most two components of Q ;
 - (c) if $x \in \text{fn}(Q)$ then x is owned by at most one component of Q ,

then Q_0 is confluent.

PROOF: Let $Q \equiv (\nu \tilde{q})(\Pi_{i \in I} T_i)$ be a derivative of Q_0 . It is easy to confirm that

- i. if $x \in \text{fn}(Q)$ and $x : i \in \{M, N\}$, then x does not occur unguarded in positive subject position in Q ;
- ii. for each T_i , there exists $\tilde{p}_i \in M \times N$ such that $T_i \in \mathcal{S}^{\tilde{p}_i}$ and if $(a, b) \in \tilde{p}_i$ then at most one of a and b occurs free within a T_j .

According to Proposition 3.4.9 we need to establish the following:

1. Q is τ -inert;
2. if $Q \xrightarrow{\alpha} Q_1, Q \xrightarrow{\alpha} Q_2, \alpha \in \text{Act}$, then $Q_1 \approx Q_2$;
3. if $Q \xrightarrow{\alpha} Q_1, Q \xrightarrow{\beta} Q_2, \alpha, \beta \in \text{Act}, \alpha \bowtie \beta$, then there exist Q'_1, Q'_2 such $Q_1 \xrightarrow{\beta} Q'_1$ and $Q_2 \xrightarrow{\alpha} Q'_2 \approx Q'_1$.

Proof of 1: Suppose $Q \xrightarrow{\tau} Q'$. Two possibilities exist:

- $Q' = (\nu \tilde{q})(T'_m \mid \Pi_{i \neq m} T_i)$, where $T_m \xrightarrow{\tau} T'_m$.
- $Q' = (\nu \tilde{q})(T'_m \mid T'_n \mid \Pi_{i \neq m, n} T_i)$, where $T_m \xrightarrow{\alpha} T'_m$ and $T_n \xrightarrow{\bar{\alpha}} T'_n$.

In the former case, it is easy to see by the ‘partial confluence’ property satisfied by T_m that $T_m \approx T'_m$ and thus $Q \approx Q'$. The proof of the latter case is similar to the proof of Lemma 3.5.2. Attention is necessary as the components here are not confluent. However, observation (ii) above allows us to conclude the required result.

Proof of 2: Suppose $Q \xrightarrow{\alpha} Q_1, Q \xrightarrow{\alpha} Q_2$. Since $\text{subj}(\alpha)$ occurs free in Q , by assumption 2(b), it occurs in subject position in exactly one component of Q . Hence there exists k such that

$$T_k \xrightarrow{\alpha} T'_k, T_k \xrightarrow{\alpha} T''_k$$

and

$$Q_1 \equiv (\nu \tilde{q}')(T'_k \mid \Pi_{i \neq k} T_i), \quad Q_2 \equiv (\nu \tilde{q}'')(T''_k \mid \Pi_{i \neq k} T_i).$$

Since \mathcal{S} is an L_{MN} -confluent partition, $T'_k \approx T''_k$ and so $Q_1 \approx Q_2$.

Proof of 3: Suppose

$$Q \xrightarrow{\alpha} Q_1 = (\nu \tilde{u})(T'_k \mid \Pi_{i \neq k} T_i)$$

where $T_k \xrightarrow{\alpha} T'_k$, and

$$Q \xrightarrow{\beta} Q_2 = (\nu \tilde{u})(T'_l \mid \Pi_{i \neq l} T_i)$$

where $T_l \xrightarrow{\beta} T'_l$. If $k \neq l$ then clearly

$$Q_1 \xrightarrow{\beta} Q' = (\nu \tilde{u})(T'_k \mid T'_l \mid \Pi_{i \neq k, l} T_i)$$

and

$$Q_2 \xrightarrow{\alpha} Q' = (\nu \tilde{u})(T'_k \mid T'_l \mid \Pi_{i \neq k, l} T_i).$$

Otherwise, suppose $k = l$. By observation (i), $\alpha, \beta \notin N^+ \cup M^+$. So, by the partial confluence property satisfied by the components, $T'_k \xrightarrow{\beta} T'$, $T'_l \xrightarrow{\alpha} T'' \approx T'$ and

$$Q_1 \xrightarrow{\beta} Q'_1 = (\nu \tilde{u})(T' \mid \Pi_{i \neq k} T_i), \quad Q_2 \xrightarrow{\alpha} Q'_2 = (\nu \tilde{u})(T'' \mid \Pi_{i \neq k} T_i)$$

where $Q'_1 \approx Q'_2$ as required.

This completes the proof. \square

3.6 \asymp -confluence

In this section we consider alternative definitions of confluence in terms of equivalences other than observation equivalence and investigate their relation to the original notion. The results obtained are similar to those of Section 3.3, and they will be useful in later chapters where branching bisimilarity is the required notion of equivalence for the systems under study. The results we present are not specific to the π_b -calculus but can be confirmed for other process calculi. The definition of \asymp -confluence follows.

Definition 3.6.1 An agent P is \asymp -confluent if, for every derivative Q of P , and for all actions α and β such that $\alpha \bowtie \beta$, the following diagrams can be completed, given transitions from Q to Q_1 and Q_2 .

$$\begin{array}{ccc} Q & \xrightarrow{\alpha} & Q_1 \\ \alpha \downarrow & & \downarrow \\ Q_2 & \Rightarrow & Q'_2 \asymp Q'_1 \end{array} \qquad \begin{array}{ccc} Q & \xrightarrow{\alpha} & Q_1 \\ \beta \downarrow & & \widehat{\beta|_{\alpha}} \downarrow \\ Q_2 & \xRightarrow{\widehat{\alpha|\beta}} & Q'_2 \asymp Q'_1 \end{array}$$

The notion of \asymp -confluence is well-behaved:

Proposition 3.6.2 \asymp -confluence is closed under derivation. □

It is easy to see that a \asymp -confluent agent is \asymp -inert.

Proposition 3.6.3 If P is \asymp -confluent then P is \asymp -inert.

PROOF: Similar to that of Proposition 3.4.6. □

Thus we may deduce that \approx_{tr} -confluence coincides with \simeq -confluence.

Proposition 3.6.4 P is \approx_{tr} -confluent iff P is \simeq -confluent.

PROOF: By Proposition 3.3.4, \approx_{tr} coincides with \simeq for $\tau_{\approx_{tr}}$ -inert agents. Since a \approx_{tr} -confluent and thus a \simeq -confluent agent is $\tau_{\approx_{tr}}$ -inert, it is easy to see that \approx_{tr} -confluence and \simeq -confluence define identical notions. □

Finally, we observe that confluence is preserved by $=_{wc}$ -confluence and thus by all the finer equivalences. □

Proposition 3.6.5 If P is confluent and $P =_{wc} Q$, then Q is confluent.

PROOF: First we show that $=_{wc}$ preserves τ -inertness. For suppose P is τ -inert and $P =_{wc} Q$, where (S_1, S_2) is a weak coupled simulation relating P and Q . Further, suppose that $Q \Rightarrow Q'$. Then there exists P' such that $P \Rightarrow P'$, and $P' S_2 Q'$. By definition of $=_{wc}$, $P' \Rightarrow P''$ where $P'' S_1 Q'$. Thus, since by τ -inertness $P' =_{wc} P''$, $P' =_{wc} Q'$ and $Q =_{wc} Q'$ as required.

So suppose P is confluent and $P =_{wc} Q$ where (S_1, S_2) is a weak coupled simulation relating P and Q . Further, suppose that $Q \xrightarrow{\alpha} Q_1$ and $Q \xrightarrow{\alpha} Q_2$. Then there exist P'_1, P'_2 such that $P \xrightarrow{\alpha} P'_1$, $P \xrightarrow{\alpha} P'_2$ and $P'_1 S_2 Q_1$ and $P'_2 S_2 Q_2$. By confluence, $P'_1 =_{wc} P'_2$. Furthermore, by the definition of $=_{wc}$, $P'_1 \Rightarrow P'_1$ and $P'_2 \Rightarrow P'_2$ such that $P'_1 S_1 Q_1$ and $P'_2 S_1 Q_2$. By confluence, we also have that $P'_1 =_{wc} P'_1$ and $P'_2 =_{wc} P'_2$. Thus, $P'_1 =_{wc} Q_1$ and $P'_2 =_{wc} Q_2$, which implies that $Q_1 =_{wc} Q_2$ as required. Finally suppose that $Q \xrightarrow{\alpha} Q_1$ and $Q \xrightarrow{\beta} Q_2$. Then, by $=_{wc}$, there exist P'_1, P'_2 such that $P \xrightarrow{\alpha} P'_1 \Rightarrow P'_1$, $P \xrightarrow{\beta} P'_2 \Rightarrow P'_2$ and $P'_1 S_2 Q_1$, $P'_2 S_2 Q_2$, $P'_1 S_1 Q_1$ and $P'_2 S_1 Q_2$. By confluence $P'_2 \xrightarrow{\alpha} P''_2$ and $P'_1 \xrightarrow{\beta} P''_1$, $P''_1 =_{wc} P''_2$. Thus $Q_1 \xrightarrow{\beta} Q'_1 S_1 P''_1$ and $Q_2 \xrightarrow{\alpha} Q'_2 S_1 P''_2$. By the definition of $=_{wc}$, $P'_1 \Rightarrow P''_1$ and $P'_2 \Rightarrow P''_2$ such that $P''_1 S_2 Q'_1$ and $P''_2 S_2 Q'_2$. Thus, since by τ -inertness $Q'_1 =_{wc} Q''_1$ and $Q'_2 =_{wc} Q''_2$, we have that $P'_1 =_{wc} Q'_1$ and $P'_2 =_{wc} Q'_2$, which implies that $Q'_1 =_{wc} Q'_2$ as required. This completes the proof. □

3.7 Divergence

In this section, we define a notion of confluence which is sensitive to divergence and we develop some of its theory. It is observed that it satisfies modified versions of the confluence properties of Section 3.3. A useful observation is that the new setting allows a simple characterization of confluence, in the vein of Proposition 3.4.9, without the additional assumption of full convergence. Although we concentrate only on proving results that will be required in later sections, we believe that the results of Sections 3.3 and 3.4 may be easily modified and proved in this setting.

The new notion of confluence is based on d-bisimilarity. The following gives a name to an interesting part of the transition system of an agent.

Definition 3.7.1 The *convergent core* of an agent P is

$$cc(P) = \{Q \mid \text{for some } s \in Act^*, P \downarrow s \text{ and } P \xRightarrow{s} Q\}.$$

Now we can define the notion of (weak) determinacy sensitive to divergence.

Definition 3.7.2 P is *d-determinate* if, for every $Q \in cc(P)$ and for all $\alpha \in Act^+$, whenever $Q \downarrow \alpha$, $Q \xrightarrow{\alpha} Q'$ and $Q \xRightarrow{\hat{\alpha}} Q''$ then $Q' \approx_1 Q''$.

Similarly, the notion of divergence-sensitive (weak) confluence, d-confluence, is the following.

Definition 3.7.3 A process P is *d-confluent* if, for every $Q \in cc(P)$ and for all $\alpha, \beta, \gamma \in Act^+$ such that $\beta \bowtie \gamma$, if $Q \downarrow \alpha$, $Q \downarrow \beta(\gamma|\beta)$ then $Q \downarrow \gamma(\beta|\gamma)$ and, given transitions from Q to Q_1 and Q_2 , the following diagrams can be completed.

$$\begin{array}{ccc} Q & \xrightarrow{\alpha} & Q_1 \\ \alpha \downarrow & & \downarrow \\ Q_2 & \Rightarrow & Q'_2 \approx_1 Q'_1 \end{array} \qquad \begin{array}{ccc} Q & \xrightarrow{\beta} & Q_1 \\ \gamma \downarrow & & \gamma \widehat{|\beta} \downarrow \\ Q_2 & \xRightarrow{\widehat{\beta|\gamma}} & Q'_2 \approx_1 Q'_1 \end{array}$$

Consider $A \stackrel{\text{def}}{=} \tau.A + a.0 + b.0$. Clearly, A is d-confluent since $cc(A) = \emptyset$ but A is not confluent since $A \xrightarrow{a} 0$ and $A \xrightarrow{b} 0$ and the matching actions can not be performed. By contrast, the agent $B = a.b.\Omega + a.b.0$ is confluent but not d-confluent since $b.\Omega \not\approx_1 b.0$. So, confluence and d-confluence are incomparable – neither implies the other. However, it is easy to see that the two notions of confluence, and for that matter the two notions of bisimilarity, coincide in the setting of convergent agents.

Proposition 3.7.4 Let P, Q be fully convergent agents. Then

- (a) $P \approx Q$ iff $P \approx_1 Q$, and

(b) P is d-confluent iff P is confluent. \square

Even though confluence and d-confluence are incomparable, we may expect that some properties of confluence may also be satisfied by d-confluence. Moreover, there may exist additional conditions under which our previous results can be extended to the divergence-sensitive setting. We investigate this possibility. First, we note that as in the case of confluence, d-confluence satisfies the following property.

Proposition 3.7.5 The relation d-confluence is preserved by d-bisimilarity. \square

However, it is easy to see that d-confluence is not closed under derivation. For example, the process $S = a.(b.0 + a.0) + \tau.S$ being divergent, is d-confluent, but $S \xrightarrow{a} b.0 + a.0$ which is not d-confluent. This is not surprising as d-confluence only requires of an agent that it be well-behaved for as long as it is convergent. Nonetheless, the following can easily be seen to hold:

Proposition 3.7.6 Let P be a d-confluent agent. Then, for all $Q \in \text{cc}(P)$, Q is d-confluent. \square

Moreover, d-bisimilarity of d-confluent agents is not preserved under silent actions. For example, if $P = \tau.0 + \tau.P$, then $P \xrightarrow{\tau} P' = 0 \not\approx_1 P$. Instead we have the following result:

Proposition 3.7.7 If P is d-confluent, $P \downarrow$ and $P \Rightarrow P'$ then $P \approx_1 P'$.

PROOF: With the additional assumption of $P \downarrow$ the proof is similar to that of Proposition 3.4.6. \square

This suggests that d-confluence implies a different notion of τ -inertness.

Definition 3.7.8 An agent P is η -inert if, for all $Q \in \text{cc}(P)$, whenever $Q \Rightarrow Q'$ then $Q \approx_1 Q'$.

By Propositions 3.7.6 and 3.7.7 we have the following corollary.

Corollary 3.7.9 If P is d-confluent then P is η -inert. \square

We proceed by presenting an alternative characterization of d-confluence that considers single steps for the initial derivations leading to Q_1 and Q_2 in the diagram of Definition 3.7.3. Considering the fact that d-confluence restricts attention on the convergent core of a process we might expect that unlike Proposition 3.4.9, in this setting the characterization is not restricted to fully convergent agents. This indeed turns out to be the case.

Proposition 3.7.10 P is d-confluent iff it is η -inert and for all $Q \in \text{cc}(P)$, and for all $\alpha, \beta, \gamma \in \text{Act}$ the following hold.

- If $Q \downarrow \alpha$, $Q \xrightarrow{\alpha} Q_1$, $Q \xrightarrow{\alpha} Q_2$ then $Q_1 \approx_1 Q_2$.
- If $\beta \bowtie \gamma$, $Q \downarrow \beta(\gamma|\beta)$ then $Q \downarrow \gamma(\beta|\gamma)$ and, given transitions to Q_1 and Q_2 , the following diagrams can be completed.

$$\begin{array}{ccc} Q & \xrightarrow{\alpha} & Q_1 \\ \alpha \downarrow & & \downarrow \\ Q_2 & \Rightarrow & Q'_2 \approx_1 Q'_1 \end{array} \qquad \begin{array}{ccc} Q & \xrightarrow{\beta} & Q_1 \\ \gamma \downarrow & & \gamma|\beta \downarrow \\ Q_2 & \xRightarrow{\beta|\gamma} & Q'_2 \approx_1 Q'_1 \end{array}$$

PROOF: It is easy to see that if P is confluent then the diagrams above can be completed. Moreover, by Proposition 3.7.9, P is η -inert.

To prove the converse suppose P is an agent satisfying the properties of the proposition. Let $Q \in \text{cc}(P)$ and suppose

$$Q \xrightarrow{\alpha} Q_1 \text{ and } Q \Rightarrow Q'_2 \xrightarrow{\beta} Q_2.$$

Since $Q \in \text{cc}(P)$ it must be that Q is η -inert. Thus $Q \approx_1 Q'_2$ and

$$Q'_2 \Rightarrow Q'_1 \xrightarrow{\alpha} Q''_1 \approx_1 Q_1,$$

where $Q'_2 \approx_1 Q'_1$. In addition, since $Q \in \text{cc}(P)$, $Q \downarrow$ and so $Q'_1 \downarrow$. So, there exists Q''_1 such that

$$Q'_1 \Rightarrow Q''_1 \not\rightarrow$$

By η -inertness, $Q'_1 \approx_1 Q''_1$. Hence $Q''_1 \xrightarrow{\alpha} Q_3 \approx_1 Q_1$ and $Q''_1 \xrightarrow{\beta} Q_4 \approx_1 Q_2$. By the assumption of the proposition,

$$Q_3 \xRightarrow{\beta} Q'_3 \text{ and } Q_4 \xRightarrow{\alpha} Q'_4 \approx_1 Q'_3.$$

Hence

$$Q_1 \xRightarrow{\beta} Q'_1 \approx_1 Q'_3 \text{ and } Q_2 \xRightarrow{\alpha} Q'_2 \approx_1 Q'_3$$

and so $Q'_1 \approx_1 Q'_2$ as required.

The case $\alpha = \beta$ follows similarly. \square

Before we proceed with the development of the theory of d-confluence, we observe that it coincides with the notion of db-confluence, the divergence-sensitive notion of confluence based on db-bisimilarity, defined below.

Definition 3.7.11 A process P is *db-confluent* if, for every $Q \in \text{cc}(P)$ and for all $\alpha, \beta, \gamma \in \text{Act}^+$ such that $\beta \bowtie \gamma$, if $Q \downarrow \alpha$, $Q \downarrow \beta(\gamma|\beta)$ then $Q \downarrow \gamma(\beta|\gamma)$ and, given transitions from Q to Q_1 and Q_2 , the following diagrams can be completed.

$$\begin{array}{ccc}
 Q & \xrightarrow{\alpha} & Q_1 \\
 \alpha \downarrow & & \downarrow \\
 Q_2 & \Rightarrow & Q'_2 \simeq_1 Q'_1
 \end{array}
 \qquad
 \begin{array}{ccc}
 Q & \xrightarrow{\beta} & Q_1 \\
 \gamma \downarrow & & \widehat{\gamma\beta} \downarrow \\
 Q_2 & \xRightarrow{\widehat{\beta\gamma}} & Q'_2 \simeq_1 Q'_1
 \end{array}$$

We have:

Proposition 3.7.12 P is d-confluent iff P is db-confluent.

PROOF: The proof is similar to that of Proposition 3.6.4 and is omitted. \square

One might expect that the analogue of Lemma 3.5.2 can be obtained for d-confluent agents. So consider a friendly agent

$$S \stackrel{\text{def}}{=} (\nu \tilde{z})(\Pi P_i \mid \Pi! C_j)$$

where components P_1 and P_2 are d-confluent and suppose $P_1 \xrightarrow{\alpha} P'_1$, $P_2 \xrightarrow{\bar{\alpha}} P'_2$ where $\alpha = a\langle \tilde{x} \rangle$, $\bar{\alpha} = (\nu \tilde{u})\bar{a}\langle \tilde{x} \rangle$. Further, let

$$S' = (\nu \tilde{z}\tilde{u})(P'_1 \mid P'_2 \mid \Pi_{i \neq 1,2} P_i \mid \Pi! C_j).$$

We see that if $P_1 \uparrow$ and $P'_1 \downarrow$ then, although $S \uparrow$, it is possible that $S' \downarrow$ and thus $S \not\approx_1 S'$. So, care is required in handling the possibility that a divergent system may evolve to a convergent agent. The following refinement handles this in a satisfactory way.

Definition 3.7.13 An agent P is *fully d-confluent* if it is d-confluent and, for all derivatives Q of P and $\alpha \in \text{Act}^+$, if $Q \uparrow \alpha$, $Q \xrightarrow{\alpha} Q'$ then $Q' \uparrow$.

Thus if a derivative of a fully d-confluent agent diverges on an action, it may not perform that action and reach a convergent state. We may now obtain the analogues of Lemmas 3.5.2 and 3.5.3. First we have a result concerning the divergent behaviour of an effective friendly agent.

Lemma 3.7.14 Suppose $S \stackrel{\text{def}}{=} (\nu \tilde{z})(\Pi P_i \mid \Pi! (\nu \tilde{y})\alpha_j. Q_j)$ is an effective friendly agent where for all i, j , P_i and Q_j are fully d-confluent. Then the following hold:

- a. If $S \uparrow$ and $S \xrightarrow{\tau} S'$ then $S' \uparrow$.
- b. If $S \uparrow \beta$ and $S \xrightarrow{\tau} S'$ then $S' \uparrow \beta$.
- c. If $S \uparrow$ and $S \xrightarrow{\beta} S'$ then $S' \uparrow$.
- d. If $S \uparrow \beta$ and $S \xrightarrow{\beta} S'$ then $S' \uparrow$.

PROOF: First we consider (a) and (b). The following possibilities exist for $S \xrightarrow{\tau} S'$:

- $S' = (\nu\tilde{p})(P'_1 \mid P_2 \mid \dots \mid P_n \mid \Pi!C_j), P_1 \xrightarrow{\tau} P'_1$.
- $S' = (\nu\tilde{p}\tilde{u})(P'_1 \mid P_{n+1} \mid P_2 \mid \dots \mid P_n \mid \Pi!C_j), P_1 \xrightarrow{\alpha} P'_1, !C_i \xrightarrow{\bar{\alpha}} !C_i \mid P_{n+1}$, where $\alpha \text{ comp } \bar{\alpha}, \tilde{u} = \text{bn}(\alpha) \cup \text{bn}(\bar{\alpha})$.
- $S' = (\nu\tilde{p}\tilde{u})(P'_1 \mid P'_2 \mid P_3 \mid \dots \mid P_n \mid \Pi!C_j), P_1 \xrightarrow{\alpha} P'_1$ and $P_2 \xrightarrow{\bar{\alpha}} P'_2$, where $\alpha \text{ comp } \bar{\alpha}, \tilde{u} = \text{bn}(\alpha) \cup \text{bn}(\bar{\alpha})$.

We consider the last possibility. The other two are easier. Let \mathcal{R}_1 be the following relation:

$$\begin{aligned} \mathcal{R}_1 = \{ & (S_1, S_2) \mid S_1 = (\nu\tilde{p})(P_1 \mid P_2 \mid P_3 \mid \dots \mid P_n \mid \Pi!C_j), \\ & S_2 = (\nu\tilde{p}\tilde{u})(P'_1 \mid P'_2 \mid P_3 \mid \dots \mid P_n \mid \Pi!C_j), \\ & P_1 \xrightarrow{\alpha} \approx_! P'_1, P_2 \xrightarrow{\bar{\alpha}} \approx_! P'_2, \alpha = a\langle\tilde{x}\rangle, \bar{\alpha} = (\nu\tilde{u})\bar{a}\langle\tilde{x}\rangle, \\ & \text{and } S_1 \text{ satisfies the assumptions of the lemma} \} \end{aligned}$$

Let $(S_1, S_2) \in \mathcal{R}_1$ where S_1 and S_2 are as above and $S_2 \downarrow$. Suppose $S_1 \xrightarrow{\tau} S'_1 = (\nu\tilde{p}')(T_1 \mid \dots \mid T_n \mid \Pi!C_j)$. We claim that there exists S'_2 such that $S_2 \Rightarrow S'_2$ and $(S'_1, S'_2) \in \mathcal{R}_1 \cup \approx_!$. Note that since $S_2 \downarrow, P'_1 \downarrow, P'_2 \downarrow$ and $P_i \downarrow$ for all $i > 2$. Thus by full d-confluence, $P_1 \downarrow \alpha$ and $P_2 \downarrow \bar{\alpha}$. The following possibilities exist:

- $S_1 \xrightarrow{\tau} S'_1 = (\nu\tilde{p}')(T_k \mid \Pi_{i \neq k} P_i \mid \Pi!C_j)$ where $P_k \xrightarrow{\tau} T_k$. By full d-confluence and since $P_k \downarrow, P_k \approx_! T_k$. Thus $(S'_1, S_2) \in \mathcal{R}_1$ as required.
- $S_1 \xrightarrow{\tau} S'_1 = (\nu\tilde{p}\tilde{w})(T_k \mid T_l \mid \Pi_{i \neq k,l} P_i \mid \Pi!C_j)$ where $P_k \xrightarrow{\sigma} T_k, P_l \xrightarrow{\bar{\sigma}} T_l, \sigma \text{ comp } \bar{\sigma}, \tilde{w} = \text{bn}(\sigma) \cup \text{bn}(\bar{\sigma})$. There exist three cases.

1. First suppose $k \neq 1, 2$ and $l \neq 1, 2$. Then it is easy to see that $S_2 \xrightarrow{\tau} S'_2$ where $(S'_1, S'_2) \in \mathcal{R}_1$ as required.
2. Otherwise, suppose $\text{subj}(\alpha) = \text{subj}(\sigma)$. Since S_1 is an effective friendly system, $\text{subj}(\alpha)$ is uniquely borne and uniquely controlled in S_1 . Thus $k = 1$ and $l = 2$ and, by the consistency of the components, $\bar{\alpha} = \bar{\sigma}$. Consequently, $\alpha = \sigma$ and, by d-confluence, $T_1 \approx_! P'_1$ and $T_2 \approx_! P'_2$. Thus $S'_1 \approx_! S_2$ as required.
3. Finally suppose $\text{subj}(\alpha) \neq \text{subj}(\sigma)$ and $k = 1, l = 2$. The cases $k = 1, l \neq 2$ and $k \neq 1, l = 2$ are similar. It is easy to see that $P_1 \downarrow \alpha\sigma, P_2 \downarrow \bar{\alpha}\bar{\sigma}$ as otherwise by full d-confluence $P'_1 \uparrow \sigma, P_2 \uparrow \bar{\sigma}$ and $S'_2 \uparrow$ which is a contradiction. Hence, by d-confluence, $P_1 \downarrow \sigma\alpha, P_2 \downarrow \bar{\sigma}\bar{\alpha}$ and

$$P'_1 \xrightarrow{\sigma} T'_1, T_1 \xrightarrow{\alpha} \approx_! T'_1,$$

$$P'_1 \xRightarrow{\bar{\sigma}} T'_2, T_2 \xRightarrow{\bar{\alpha}} \approx_1 T'_2.$$

This implies that

$$S_2 \Rightarrow S'_2 = (\nu \tilde{p}' \tilde{u})(T'_1 | T'_2 | P_3 | \dots | P_n | \Pi! C_j)$$

and $(S_2, S'_2) \in \mathcal{R}_1$ as required.

We now proceed to prove (a). Let $(S, S') \in \mathcal{R}_1$ and suppose $S \uparrow$. If either of $P_1 \uparrow \alpha$, $P_2 \uparrow \bar{\alpha}$, $P_i \uparrow$ for $i > 2$ then $S' \uparrow$ (in the case of $P_1 \uparrow \alpha$ (resp. $P_2 \uparrow \bar{\alpha}$) because, by full d-confluence, $P'_1 \uparrow$ (resp. $P'_2 \uparrow$)). So assume $P_1 \downarrow \alpha$, $P_2 \downarrow \bar{\alpha}$, $P_i \downarrow$ for $i > 2$. Since $S \uparrow$, for $S_m = (\nu \tilde{p} \tilde{v}_1 \dots \tilde{v}_m)(\Pi_{i \in I_m} P_i | \Pi! C_j)$, either

i. $S = S_0 \xrightarrow{\tau} S_1 \xrightarrow{\tau} \dots \xrightarrow{\tau} S_k$ with $P_i \downarrow$, for all $i \in I_m$ and $m < k$ and $P_i \uparrow$ for some $i \in I_k$, or

ii. $S = S_0 \xrightarrow{\tau} S_1 \xrightarrow{\tau} \dots$ with $P_i \downarrow$, for all $i \in I_m$ and $m < \omega$.

First consider (i) and suppose $S' \downarrow$. We will derive a contradiction. Since $P_1 \downarrow \alpha$, $P_2 \downarrow \bar{\alpha}$ and $P_i \downarrow$ for all $i \in I_m$ and $m < k$, by repeated application of the previous observation, $S' \Rightarrow S'_1 \Rightarrow \dots \Rightarrow S'_{k-1}$ where for $m < k$, $(S_m, S'_m) \in \mathcal{R}_1 \cup \approx_1$. Now consider the transition $S_{k-1} \xrightarrow{\tau} S_k$. If $S_{k-1} \approx_1 S'_{k-1}$ then we are done. Otherwise, suppose that $S_{k-1} = (\nu \tilde{p} \tilde{v}_1 \dots \tilde{v}_{k-1})(P_1 | \dots | P_n | \Pi! C_j)$ and $S_k = (\nu \tilde{p} \tilde{v}_1 \dots \tilde{v}_k)(T_1 | \dots | T_n | \Pi! C_j)$ where $T_l \uparrow$. Since $P_i \downarrow$ for all i and $T_l \uparrow$ this implies that the transition is a communication between two of the components of S_{k-1} . That is, for some $\sigma, \bar{\sigma} \neq \tau$, $\sigma \text{ comp } \bar{\sigma}$, $P_l \xrightarrow{\sigma} T_l$ and $P_m \xrightarrow{\bar{\sigma}} T_m$ and $S_k = (\nu \tilde{p} \tilde{v}_1 \dots \tilde{v}_k)(T_l | T_m | \Pi_{i \neq l, m} P_i | \Pi! C_j)$. By Clauses (2) and (3) above we may see that either $S_k \approx_1 S'_{k-1}$, or there is a transition $S'_{k-1} \xRightarrow{\tau} S'_k$ such that $(S_k, S'_k) \in \mathcal{R}_1$. If $S_k \approx_1 S'_{k-1}$ then clearly $S'_k \uparrow$. Otherwise, by construction of \mathcal{R}_1 , since $T_l \uparrow$, a component of S'_k is also divergent and thus $S'_k \uparrow$. This contradicts the assumption that $S' \downarrow$. Thus $S' \uparrow$ as required.

Alternatively, if (ii) holds, then since $P_i \downarrow$ for all $i \in I_m$ and $m < \omega$, the diverging computation contains infinitely many communications between the components of the system. By repeated application of the argument above it follows that there is a diverging computation from S' . Hence $S' \uparrow$. This completes the proof of Property (a).

Next consider (b). Suppose $(S, S') \in \mathcal{R}_1$ and $S \uparrow \beta$ for some $\beta \in \text{Act}$. We will prove that $S' \uparrow \beta$. If $S \uparrow$ then the result follows from (a). So suppose $S \downarrow \Rightarrow R \xrightarrow{\beta} Q \uparrow$ where $R = (\nu \tilde{p} \tilde{w})(\Pi T_i | \Pi! C_j)$. If $S' \uparrow$ then we are done, otherwise by the observation above, $S' \Rightarrow R' \downarrow$ where $R' = (\nu \tilde{p} \tilde{w} \tilde{u})(T'_1 | T'_2 | \Pi_{i \neq 1, 2} T_i | \Pi! C_j)$, $T_1 \xRightarrow{\alpha} \approx_1 T'_1$ and

$T_2 \xrightarrow{\bar{\alpha}} \approx_1 T'_2$ for some $\alpha = a\langle\tilde{x}\rangle$, $\bar{\alpha} = (\nu\tilde{u})\bar{a}\langle\tilde{x}\rangle$. Since S is an effective friendly system, $\text{subj}(\beta)$ is owned by exactly one component of R . So suppose $T_i \xrightarrow{\beta} U_i$. There exist two cases.

1. Suppose $i \neq 1, 2$. Then $R' \xrightarrow{\tau} Q'$ and $(Q, Q') \in \mathcal{R}_1$. Thus by (a), $Q' \uparrow$ as required.
2. Suppose $i = 1$ (the case $i = 2$ is identical). First we note that $\text{subj}(\alpha) \neq \text{subj}(\beta)$ as $\text{subj}(\alpha)$ is borne and controlled within R . If $T_1 \uparrow \alpha\beta$ then by full d-confluence, $T'_1 \uparrow \beta$. Hence $R' \uparrow \beta$ and so $S' \uparrow \beta$. Otherwise, if $T_1 \downarrow \alpha\beta$ then by d-confluence $T'_1 \xRightarrow{\beta} U'_1$, $U_1 \xRightarrow{\alpha} \approx_1 U'_1$ and $R' \xrightarrow{\beta} Q' = (\nu\tilde{p}\tilde{w}\tilde{v})(U'_1 \mid T'_2 \mid \Pi_{i \neq 1,2} T_i \mid \Pi! C_j)$, where $(Q, Q') \in \mathcal{R}_1$. Hence by part(a), since $Q \uparrow$, we have that $Q' \uparrow$ and $Q' \uparrow \beta$ as required.

The proofs of (c) and (d) follow similarly. They involve considering the relation

$$\begin{aligned} \mathcal{R}_2 = \{ & (S_1, S_2) \mid S_1 = (\nu\tilde{p})(P_1 \mid P_2 \mid \dots \mid P_n \mid \Pi! C_j), \\ & S_2 = (\nu\tilde{p} - \tilde{u})(P'_1 \mid P_2 \mid \dots \mid P_n \mid \Pi! C_j), \\ & P_1 \xRightarrow{\beta} \approx_1 P'_1, \tilde{u} = \text{bn}(\beta) \\ & \text{and } S_1 \text{ satisfies the assumptions of the lemma} \} \end{aligned}$$

and establishing that, if $(S_1, S_2) \in \mathcal{R}_2$ and $S_1 \xrightarrow{\tau} S'_1$, then $S_2 \Rightarrow S'_2$ such that $(S'_1, S'_2) \in \mathcal{R}_2 \cup \approx_1$. This completes the proof. \square

Lemma 3.7.15 Suppose $S \stackrel{\text{def}}{=} (\nu\tilde{z})(\Pi P_i \mid \Pi! C_j)$ is a friendly agent where components P_1 and P_2 are fully d-confluent and $S \downarrow$. Suppose $P_1 \xRightarrow{\alpha} \approx_1 P'_1$, $P_2 \xRightarrow{\bar{\alpha}} \approx_1 P'_2$ where $\alpha = a\langle\tilde{x}\rangle$, $\bar{\alpha} = (\nu\tilde{u})\bar{a}\langle\tilde{x}\rangle$. Let $S' = (\nu\tilde{z}\tilde{u})(P'_1 \mid P'_2 \mid \Pi_{i \neq 1,2} P_i \mid \Pi! C_j)$. Then, assuming that $a : A$ is persistently managed in S , $S \approx_1 S'$.

PROOF: Let $(S, S') \in \mathcal{B}$ if $S \stackrel{\text{def}}{=} (\nu\tilde{z})(\Pi P_i \mid \Pi! C_j)$ and $S' = (\nu\tilde{z}\tilde{u})(P'_1 \mid P'_2 \mid \Pi_{i \neq 1,2} P_i \mid \Pi! C_j)$, where $S \downarrow$, P_1 and P_2 are fully d-confluent, $P_1 \xRightarrow{\alpha} \approx_1 P'_1$, $P_2 \xRightarrow{\bar{\alpha}} \approx_1 P'_2$, $\alpha = a\langle\tilde{x}\rangle$, $\bar{\alpha} = (\nu\tilde{u})\bar{a}\langle\tilde{x}\rangle$, and a is persistently managed in S . We will show that $\mathcal{B} \cup \approx_1$ is a d-bisimulation. So suppose $(S, S') \in \mathcal{B}$ where S and S' are as above.

Suppose $S' \downarrow \beta$ and $S' \xrightarrow{\beta} T'$. Then since $S \Rightarrow T'' \approx_1 S'$, by Lemma 3.7.14(b), $S \downarrow \beta$. Further, by definition of \approx_1 , it is straightforward to find T such that $T'' \xRightarrow{\beta} T \approx_1 T'$.

Suppose $S \downarrow \beta$, $S \xrightarrow{\beta} T$. Then clearly $S' \downarrow \beta$. So, it remains to show that $S' \xRightarrow{\beta} T'$ where $(T, T') \in \mathcal{B} \cup \approx_1$. Note that since a is managed in S , and it occurs in both P_1 and P_2 then it must occur bound in S and so $a \neq \text{subj}(\beta)$. Various possibilities exist.

- $T \equiv (\nu \tilde{z} - \tilde{v})(Q_1 \mid \Pi_{i \neq 1} P_i \mid \Pi! C_j)$ where $\text{bn}(\beta) = \tilde{v}$ and $P_1 \xrightarrow{\beta} Q_1$. Since $S \downarrow \beta$ then it must be that $P_1 \downarrow \beta$. So, by full d-confluence, $P'_1 \xrightarrow{\beta \upharpoonright \alpha} Q'_1$ and $Q_1 \xrightarrow{\alpha \upharpoonright \beta} \approx_1 Q'_1 \approx_1 Q'_1$. So, $S' \xrightarrow{\beta} T' \equiv (\nu \tilde{z} \tilde{u} - \tilde{v})(Q'_1 \mid \Pi_{i \neq 1} P_i \mid \Pi! C_j)$ and $(T, T') \in \mathcal{B}$.
- If P_2 acts alone the arguments are the same as those of the previous case.
- $T \equiv (\nu \tilde{z} - \tilde{v})(P_1 \mid P_2 \mid R)$ where $\text{bn}(\beta) = \tilde{v}$ and $(\Pi_{i \neq 1,2} P_i \mid \Pi! C_j) \xrightarrow{\beta} R$. Then clearly, $S' \xrightarrow{\beta} T' \equiv (\nu \tilde{z} \tilde{u} - \tilde{v})(P'_1 \mid P'_2 \mid R)$ and $(T, T') \in \mathcal{B}$.
- If $\beta = \tau$ where one of P_1 and P_2 interacts with $(\Pi_{i \neq 1,2} P_i \mid \Pi! C_j)$ then the argument is a combination of those above. Note that since a is managed, $a \notin \text{fn}(R)$ and thus the interaction is not via a .
- If $\beta = \tau$, where P_1 and P_2 interact via a name other than a then again the case follows using arguments of the previous cases.
- Finally, suppose $\beta = \tau$ and $T' \equiv (\nu \tilde{z} - \tilde{u})(Q_1 \mid Q_2 \mid \Pi_{i \neq 1,2} P_i \mid \Pi! C_j)$ where $P_1 \xrightarrow{\sigma} Q_1$, $P_2 \xrightarrow{\bar{\sigma}} Q_2$ and $\text{subj}(\sigma) = a$. Since a is persistently managed in S , P_1 uniquely bears a and P_2 uniquely handles a . So $\sigma \in A^+$, $\bar{\sigma} \in A^-$. Recall that by the definition of a friendly agent, P_2 is consistent. Hence, $\bar{\alpha} = \bar{\sigma}$ and, as a result, $\alpha = \sigma$. So, by d-confluence, $Q_1 \approx_1 P'_1$ and $Q_2 \approx_1 P'_2$. Hence $T \approx_1 S'$.

□

Lemma 3.7.16 Suppose $S \stackrel{\text{def}}{=} (\nu \tilde{z})(\Pi_{1 \leq i \leq n} P_i \mid \Pi! C_j)$ is a friendly agent where component P_1 is fully d-confluent. Suppose $P_1 \xrightarrow{\alpha} \approx_1 P'_1, !C_k \xrightarrow{\bar{\alpha}} P_{n+1} \mid !C_k$ where $\tilde{u} = \text{bn}(\alpha) \cup \text{bn}(\bar{\alpha})$. Let $S' = (\nu \tilde{z} \tilde{u})(P'_1 \mid P_{n+1} \mid \Pi_{i \neq 1} P_i \mid \Pi! C_j)$. Then assuming that $\text{subj}(\alpha) : A$ is uniquely handled in S , $S \approx_1 S'$.

PROOF: The proof is similar to that of the previous lemma only easier since C_k is not able to act on its own and there exists exactly one communication between P_1 and C_k , namely the one performed in the transition $S \xrightarrow{\tau} S'$. □

We complete this section with the analogue of Theorem 3.5.4, asserting that an effective friendly agent composed of fully d-confluent agents is fully d-confluent.

Theorem 3.7.17 Suppose $S_0 \equiv (\nu \tilde{p})(\Pi P_i \mid \Pi! (\nu \tilde{y}) a_j. Q_j)$ is an effective friendly agent where for all i, j , P_i and Q_j are fully d-confluent. Then S_0 is fully d-confluent.

PROOF: Let $S \equiv (\nu \tilde{p})(\Pi P_i \mid \Pi! C_j)$, $S \in \text{cc}(S_0)$ be a derivative of S_0 . By Proposition 3.7.10, it suffices to prove the following:

1. If $S \xrightarrow{\tau} S'$, then $S \approx_1 S'$.
2. If $\alpha \in Act$, $S \downarrow \alpha$ and $S \xrightarrow{\alpha} S_1$, $S \xrightarrow{\alpha} S_2$, then $S_1 \approx_1 S_2$.
3. If $\alpha, \beta \in Act$, $\alpha \bowtie \beta$, $S \downarrow \alpha(\beta|\alpha)$ and $S \xrightarrow{\alpha} S_1$, $S \xrightarrow{\beta} S_2$, then $S \downarrow \beta(\alpha|\beta)$ and $S_1 \xrightarrow{\beta|\alpha} S'_1$ and $S_2 \xrightarrow{\alpha|\beta} S'_2 \approx_1 S'_1$.
4. If $S \uparrow \alpha$ and $S \xrightarrow{\alpha} S'$, then $S' \uparrow$.

Proof of Property 1: The following possibilities exist for $S \xrightarrow{\tau} S'$:

- $S' = (\nu \tilde{p})(P'_1 | P_2 | \dots | P_n | \Pi!C_j)$, $P_1 \xrightarrow{\tau} P'_1$. Since $S \downarrow$ then $P_1 \downarrow$ and so, by d-confluence, $P_1 \approx_1 P'_1$. Thus, $S \approx_1 S'$ as required.
- $S' = (\nu \tilde{p}\tilde{u})(P'_1 | P_{n+1} | P_2 | \dots | P_n | \Pi!C_j)$, $P_1 \xrightarrow{\beta} P'_1$, $!C_i \xrightarrow{\bar{\beta}} !C_i | P_{n+1}$, where $\beta \text{ comp } \bar{\beta}$ and $\tilde{u} = \text{bn}(\beta) \cup \text{bn}(\bar{\beta})$. By Lemma 3.7.16, we deduce that $S \approx_1 S'$ which completes the case.
- $S' = (\nu \tilde{p}\tilde{u})(P'_1 | P'_2 | P_3 | \dots | P_n | \Pi!C_j)$, $P_1 \xrightarrow{\alpha} P'_1$ and $P_2 \xrightarrow{\bar{\alpha}} P'_2$, where $\alpha \text{ comp } \bar{\alpha}$ and $\tilde{u} = \text{bn}(\alpha) \cap \text{bn}(\bar{\alpha})$. By Lemma 3.7.15, $S \approx_1 S'$ as required.

This completes the proof of Property 1.

Proof of Property 2: Suppose $S \downarrow \alpha$, $S \xrightarrow{\alpha} S_1$, $S \xrightarrow{\alpha} S_2$. Since S is an effective system, $\text{subj}(\alpha)$ is uniquely borne or uniquely handled in S , and thus there exists i such that $S_1 = (\nu \tilde{p})(P'_i | \dots)$, $S_2 = (\nu \tilde{p})(P''_i | \dots)$, where $P_i \xrightarrow{\alpha} P'_i$ and $P_i \xrightarrow{\alpha} P''_i$. Since $S \downarrow \alpha$, $P_i \downarrow \alpha$. So, by d-confluence, $P'_i \approx_1 P''_i$. Therefore $S_1 \approx_1 S_2$.

Proof of Property 3: Suppose $S \downarrow \alpha(\beta|\alpha)$ and $S \xrightarrow{\alpha} S_1$ and $S \xrightarrow{\beta} S_2$, where $\alpha, \beta \in Act$, $\alpha \bowtie \beta$,

$$S_1 = (\nu \tilde{p} - \tilde{u})(P'_m | \Pi P_i | \Pi!C_j), P_m \xrightarrow{\alpha} P'_m$$

and

$$S_2 = (\nu \tilde{p} - \tilde{v})(P'_n | \Pi P_i | \Pi!C_j), P_n \xrightarrow{\beta} P'_n.$$

If $m \neq n$ then

$$S_1 \xrightarrow{\beta} S' = (\nu \tilde{p} - \tilde{u}\tilde{v})(P'_m | P'_n | \Pi P_i | \Pi!C_j)$$

and

$$S_2 \xrightarrow{\alpha} S' = (\nu \tilde{p} - \tilde{u}\tilde{v})(P'_m | P'_n | \Pi P_i | \Pi!C_j).$$

If $m = n$ then since P_m is d-confluent and $P_m \downarrow \alpha(\beta|\alpha)$, $P'_m \xrightarrow{\beta|\alpha} P''_m$ and $P'_n \xrightarrow{\alpha|\beta} P''_n \approx_1 P'_m$. It then follows that

$$S_1 \xrightarrow{\beta|\alpha} S'_1 = (\nu \tilde{p} - \tilde{u}\tilde{v})(P''_m | \Pi P_i | \Pi!C_j)$$

and

$$S_2 \xrightarrow{\alpha|\beta} S'_2 = (\nu\tilde{p} - \tilde{u}\tilde{v})(P''_n \mid \Pi P_i \mid \Pi! C_j).$$

So $S'_1 \approx_1 S'_2$ as required. Moreover, $S \downarrow \beta(\alpha|\beta)$ for, if $S \uparrow \beta(\alpha|\beta)$, then by Lemma 3.7.14, $S'_2 \uparrow$ and $S'_1 \uparrow$ which contradicts the assumption of $S \downarrow \alpha(\beta|\alpha)$.

Proof of Property 4: This is a direct consequence of Lemma 3.7.14. For suppose $S \uparrow$ and $S \Rightarrow S'$. Then by repeated application of Clause (a) of the Lemma, $S' \uparrow$. Similarly, if $\alpha \in Act$, $S \uparrow \alpha$, $S \Rightarrow S_1 \xrightarrow{\alpha} S_2 \Rightarrow S'$, then by Clause (b), $S_1 \uparrow \alpha$, by Clause (d), $S'_2 \uparrow$ and, by Clause (a), $S' \uparrow$. This completes the proof. \square

3.8 Π -confluence

As mentioned earlier, a notion of confluence which has been studied in the context of value-passing calculi is that of Π -confluence, where Π is a partition of the set of actions of Act^+ . The requirement imposed of Π -confluent agents is that they are determinate with respect to all actions but need only satisfy the confluence diagram for actions in different blocks of the partition Π . This results in a variety of confluence notions both weaker and stronger than the original one, which may be especially useful when reasoning about non-confluent but somehow well-behaved agents. In this section we briefly review Π -confluence and its relation with τ -inertness, and impose requirements on an effective agent composed of Π -confluent agents which guarantee that it is confluent. This result will be employed in Chapter 6, where the confluence of a class of programs of a concurrent-object language will be established. We begin with the definition. We write $\alpha \Pi \beta$ if actions α and β belong to the same block of Π .

Definition 3.8.1 Let Π be a partition of Act^+ . A process P is Π -confluent if for all $Q \in cc(P)$,

1. for all α , if $Q \downarrow \alpha$, $Q \xrightarrow{\alpha} Q_1$ and $Q \xRightarrow{\alpha} Q_2$, then $Q_1 \approx_1 Q_2$, and
2. for all α, β with not $(\alpha \Pi \beta)$, if $Q \downarrow \alpha\beta$, $Q \xrightarrow{\alpha} Q_1$ and $Q \xRightarrow{\beta} Q_2$, then $Q \downarrow \beta\alpha$, $Q_1 \xRightarrow{\beta} Q'$, and $Q_2 \xRightarrow{\alpha} \approx_1 Q'$.

Further, P is *fully Π -confluent* if P is Π -confluent and for each derivative Q of P and each α , if $Q \uparrow \alpha$ and $Q \xrightarrow{\alpha} Q'$, then $Q' \uparrow$. \square

It is required of a Π -confluent agent P that for each derivative Q of P : (1) Q is determinate under actions which do not engender divergence; and (2) Q enjoys

a confluence property for actions that belong to different blocks of the partition Π , and which do not together introduce divergence. Full Π -confluence demands in addition that a divergent process may not evolve to a convergent state. Thus, taking $\Pi = \neg(\bowtie)$, Π -confluence coincides with db-confluence.

We have the following simple but important observation.

Lemma 3.8.2 Let Π be a partition of Act^+ such that $\{\tau\} \in \Pi$. Then, if P is Π -confluent, P is η -inert.

PROOF: The proof is standard. \square

Note that the result would not hold should we relax the condition on partition Π . A counter-example is given by agent $A \stackrel{\text{def}}{=} \alpha.0 + \tau.0$, which although $\{Act\}$ -confluent, is not η -inert.

Let us say that partition Π *refines* partition Π' if, for all $A \in \Pi$, there is $B \in \Pi'$ such that $A \subseteq B$. It is easy to see that following hold:

Lemma 3.8.3

1. Let Π, Π' be partitions of Act such that Π refines Π' . For all agents P , if P is Π -confluent then P is Π' -confluent.
2. For all agents P , P is d-determinate iff P is $\{\{\tau\}, Act\}$ -confluent.
3. Let $\Pi = \Pi' \cup \{\{\beta_1 \dots \beta_n\}\}$, and suppose that $\text{subj}(\beta_i) : I$ for some I . If P is a Π -confluent, I -closed agent, then P is $\Pi' \cup \{\{\beta_i\}, \{\beta_1 \dots \beta_n\}\}$ -confluent.

PROOF: The proof of Property 1 follows easily from the definitions. The proof of Property 2 is a direct consequence of the definitions and the previous lemma. Finally, Property 3 asserts that given an I -closed agent P , then P is Π -confluent for some partition Π such that all elements of I^\pm appear in Π as singletons. The proof of this is trivial. \square

The following definition, which extends the notion of consistency, is useful.

Definition 3.8.4 An agent S is Π -consistent if for all derivatives P of S , whenever $P \xrightarrow{\alpha}$ and $P \xRightarrow{\beta}$, where $\alpha \Pi \beta$, then $\alpha = \beta$.

We now have the main result:

Theorem 3.8.5 Let Π be a partition of Act^+ such that $\{\tau\} \in \Pi$ and suppose $S_0 \equiv (\nu \tilde{p})(\Pi P_i \mid \Pi!(\nu \tilde{y})a_j.Q_j)$ is an effective friendly agent where

1. for all i, j , P_i and Q_j are fully Π -confluent and Π -consistent;

2. for all derivatives S of S_0 , if $\alpha\Pi\beta$ and $\text{subj}(\bar{\alpha}) \text{subj}(\bar{\beta})$ occur free in components C and C' of S , then $C = C'$.

Then, S_0 is fully Π -confluent.

PROOF: The proof is a simple reformulation of the proof of Theorem 3.7.17. The important point to note is that, although the components are not confluent, η -inertness is maintained as the conditions ensure that the components are well-behaved: if for some component C of S_0 , $C \xrightarrow{\alpha} C_1$ and $C \xrightarrow{\beta} C_2$ where $\alpha\Pi\beta$, then there is a unique component of C' owning $\text{subj}(\bar{\alpha})$ and $\text{subj}(\bar{\beta})$, unique because the system is effective, and since C' is Π -consistent, at most one of the transitions is enabled, thus preventing violation of Π -confluence. \square

3.9 Labelled transition systems

The theory of this chapter has been developed in terms of π_v -calculus agents. An alternative approach, which can be obtained by an easy modification of the material, is in terms of labelled transition systems of the form $(\mathcal{P}, \{\xrightarrow{\ell} \mid \ell \in L\})$, where $L \subseteq \text{Act}^+$. For example consider $(\mathcal{P}_1, \mathcal{S}_1)$ and $(\mathcal{P}_2, \mathcal{S}_2)$ and suppose $P_1 \in \mathcal{P}_1$ and $P_2 \in \mathcal{P}_2$. We say that the points P_1, P_2 are bisimilar, and write $P_1 \approx P_2$ if the following hold:

- if $P_1 \xrightarrow{\alpha} Q_1 \in \mathcal{S}_1$ then $P_1 \xRightarrow{\alpha} Q_2 \in \mathcal{S}_2^*$ and $Q_1 \approx Q_2$, and
- vice versa.

Further, we say that Q is a derivative of P in $(\mathcal{P}, \mathcal{S})$ if $P, Q \in \mathcal{P}$ and there exist α_i such that $P \xrightarrow{\alpha_1} \dots \xrightarrow{\alpha_n} Q \in \mathcal{S}^*$. Building on these definitions we may define when a point P of a transition system is confluent:

Definition 3.9.1 Let $(\mathcal{P}, \mathcal{S})$ be a labelled transition system and suppose $P \in \mathcal{P}$. We say that P is *locally confluent* if for all actions α and β , $\alpha \bowtie \beta$, there exist transitions in \mathcal{S} such that the following diagrams can be completed, given transitions from P to P_1 and P_2 .

$$\begin{array}{ccc}
 P & \xrightarrow{\alpha} & P_1 \\
 \alpha \downarrow & & \downarrow \\
 P_2 & \Rightarrow & P'_2 \approx P'_1
 \end{array}
 \qquad
 \begin{array}{ccc}
 P & \xrightarrow{\alpha} & P_1 \\
 \beta \downarrow & & \downarrow \widehat{\beta[\alpha]} \\
 P_2 & \xRightarrow{\widehat{\alpha|\beta}} & P'_2 \approx P'_1
 \end{array}$$

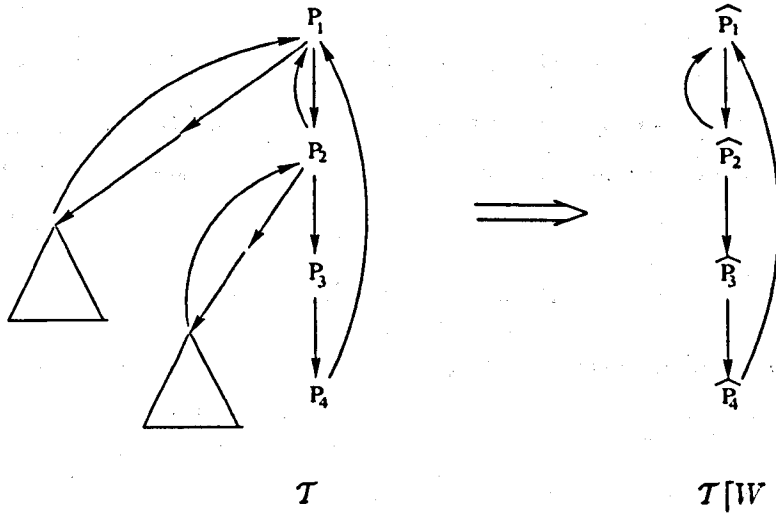
Further we say that P is *confluent* if for all derivatives Q of P , Q is locally confluent. \square

Similarly, we may modify the remaining definitions of the chapter and obtain analogous results for labelled transition systems.

In subsequent chapters we will be interested to consider in detail only parts of the transition system of π_v -calculus agents. To assist us we define the pruning operation as follows:

Definition 3.9.2 Given a labelled transition system \mathcal{T} and a subset W of its set of points we define $\mathcal{T}[W]$ to be the system obtained by removing all points not in W from \mathcal{T} and all arrows incident on such points.

The effect of this operation is illustrated in the following figure, where $W = \{P_1 \dots P_4\}$, and \hat{P} denotes the point of $P \in W$ corresponding to P in $\mathcal{T}[W]$.



We consider an example. Let P be an agent and suppose \mathcal{P} is the labelled transition system generated by P . Let R be a sort and V the following set of derivatives of P :

$$V = \{P' \mid \exists n, \mu_1 \dots \mu_n : P = P_0 \xRightarrow{\mu_1} P_1 \xRightarrow{\mu_2} \dots \xRightarrow{\mu_n} P_n = P',$$

$$\text{and, if } \mu_i = a\langle \tilde{y} \rangle \text{ and } x : R \in \tilde{y}, \text{ then } x \notin \text{fn}(P_{i-1})\}$$

The transition system $\mathcal{P}[V]$ contains the points of \mathcal{P} which are obtained via a transition involving only new R -names. We denote $\mathcal{P}[V]$ by \mathcal{P}^R and given a point Q in V we write Q^R for the point of \mathcal{P}^R corresponding to P . Note that if $Q^R \xrightarrow{\alpha} S$ then there exists Q' such that $Q \xrightarrow{\alpha} Q'$ and $Q'^R = S$. We are particularly interested in capturing conditions under which the labelled transition system defined by Q^R is confluent. We begin by formalizing when such a system is effective.

Definition 3.9.3 Let $S = (\nu \tilde{x})(\Pi P_i \mid \Pi! C_k)$ be a friendly agent, where $C_j \equiv (\nu \tilde{y}_j) \overline{a_j} \langle \tilde{x}_j \rangle . Q_j$, and $a_j : T_j$ and consider S^R . We say that S^R is *locally effective* if for all $A \neq T_j$,

1. if for some j , C_j is not A -closed, then S manages all names $x : A$, and
2. at most one component of S handles all names $x : T_j$ for all j .

Further, S^R is *effective* if, for all derivatives Q^R of S^R , Q^R is locally effective.

Thus we may obtain the analogue of Theorems 3.7.17 and 3.8.5.

Theorem 3.9.4 Suppose $S_0 \equiv (\nu \tilde{p})(\Pi P_i \mid \Pi! (\nu \tilde{y}) \alpha_j . Q_j)$ is a friendly agent where for all i, j , P_i and Q_j are fully d-confluent. Suppose S_0^R is effective. Then S_0^R is fully d-confluent.

PROOF: This is similar to the proof of Theorem 3.7.17. □

Theorem 3.9.5 Let Π be a partition of Act^+ such that $\{\tau\} \in \Pi$ and suppose $S_0 \equiv (\nu \tilde{p})(\Pi P_i \mid \Pi! (\nu \tilde{y}) \alpha_j . Q_j)$ is a friendly agent where S_0^R is effective and the following hold:

1. for all i, j , P_i and Q_j are fully Π -confluent and Π -consistent;
2. for all derivatives S of S_0 , if $\alpha \Pi \beta$ and $\text{subj}(\overline{\alpha}) \text{subj}(\overline{\beta})$ occur free in components C and C' of S , then $C = C'$.

Then, S_0^R is fully Π -confluent.

PROOF: This is similar to the proof of Theorem 3.8.5.

3.10 A Protocol Example

In this section, we illustrate the π_v -calculus and some of the notions associated with it in the verification of a protocol. In addition, we show how the theory of confluence may be employed to facilitate the correctness proof. The protocol we consider is a variant of Segall's PIF (Propagation of Information with Feedback) protocol [Seg83] and is taken from [Vaa95], where its correctness proof was established within the I/O-automata model. Its behaviour involves constructing a spanning tree of the graph connecting the nodes of a certain system and using it to compute the sum of the weights of the nodes. Although the verification methods used in the literature to reason about the algorithm allow for a representation of the creation of the spanning tree (via, for example, the use of a history variable in the proof of

[Vaa95]), the π_v -calculus provides a dynamic description of the system and thus a *direct* representation of the tree creation at the level of the system's interconnection. Supplemented with the use of sorts, this capability facilitates the proof of correctness. Furthermore, using a confluence argument we reduce the problem of establishing that all computations of the system are correct to ensuring that a single computation satisfies the specification.

3.10.1 The Algorithm

We begin with a brief, informal description of the algorithm and continue with the process-calculus description. The system consists of a collection of nodes which represent autonomous processors and a collection of links connecting the nodes which represent bidirectional communication channels via which the processors may send messages to one another. We assume that each of the nodes has an integer weight associated with it and that the graph corresponding to the network of nodes is connected.

The algorithm computes the sum of the weights of the nodes in the system as follows: a node enters the protocol when it receives a message along one of its links l . On receiving this message, it marks l and sends the value 0 along its other links. Concurrently, it may receive messages along its other links. Integer messages received in communications by a node are added to the weight of the node. As soon as the number of values received is equal to the number of the node's links, the weight of the node is sent to the node from which the first message was received. Computation is initiated by a distinguished node in the system called the root. The root enters the protocol by receiving a start message from the environment and, on completion, it sends the value it has computed, the final outcome of the algorithm, to the environment.

We now turn to giving the process-calculus description of the system. The sorting we employ makes use of the sorts L (for link), T (for tree) and N (for neighbour). The sorting λ is given by the following partial function:

$$\begin{aligned}\lambda(L) &= \{(T, N)\} \\ \lambda(T) &= \{(int)\} \\ \lambda(N) &= \{()\}\end{aligned}$$

Thus the sorting decrees that a name of sort L carries a pair of names of sorts T and N . In turn a name of sort T carries an integer value and a name of sort N carries an empty tuple.

The definition of agent $\text{Node}(\tilde{s}, v)$ representing a node with adjacent links \tilde{s} of sort L and weight v is given by the following families of equations:

$$\begin{aligned}
 \text{Node}(\tilde{s}, v) &\stackrel{\text{def}}{=} \sum_{s \in \tilde{s}} s(p, n). \text{Busy}(p, \tilde{s} - s, \tilde{s} - s, \varepsilon, \varepsilon, v) \\
 \text{Busy}(p, \tilde{s}_1, \tilde{s}_2, \tilde{x}, \tilde{y}, v) &\stackrel{\text{def}}{=} \sum_{s \in \tilde{s}_1} (\nu q, n) \bar{s}(q, n). \text{Busy}(p, \tilde{s}_1 - s, \tilde{s}_2, \tilde{x}, \tilde{y}(q, n), v) \\
 &\quad + \sum_{s \in \tilde{s}_2} s(q, n). \text{Busy}(p, \tilde{s}_1, \tilde{s}_2 - s, \tilde{x}, \tilde{y}, v) \\
 &\quad + \sum_{(q, n) \in \tilde{y}} q(v'). \text{Busy}(p, \tilde{s}_1, \tilde{s}_2, \tilde{x}, \tilde{y} - (q, n), v + v') \\
 &\quad + \sum_{(q, n) \in \tilde{y}} n. \text{Busy}(p, \tilde{s}_1, \tilde{s}_2, \tilde{x}, \tilde{y} - (q, n), v) \\
 &\quad + \sum_{x \in \tilde{x}} \bar{x}. \text{Busy}(p, \tilde{s}_1, \tilde{s}_2, \tilde{x} - x, \tilde{y}, v) \\
 \text{Busy}(p, \varepsilon, \varepsilon, \varepsilon, \varepsilon, v) &\stackrel{\text{def}}{=} \bar{p}(v). 0
 \end{aligned}$$

Thus, in its quiescent state, Node may receive a message on any of its links, consisting of a pair of names, $p : T$ and $n : N$. On doing so it discards the second name and assumes the state Busy . In this state it is responsible for using each of its remaining links to send and to receive a pair of names, as described in the first two summands. In the former case, one of the newly-created links will be used as follows: the name of sort T may be used for the receipt of an integer which will subsequently be added to the weight of the node and the name of sort N may be used in a synchronization (third and fourth summands). In the latter case the first of the names received is discarded while the second one is used by the agent to output an empty tuple (fifth summand). Once all communications are completed, the node will emit its weight via link p and become inactive.

An output action of the form $\bar{s}(p, n)$ by a node α may be thought of as a question addressed to a neighbouring node β whether it has already entered the protocol or not. The answer is provided by β by communicating via one of the names supplied. The choice of the name takes place as follows: if β is in the quiescent state then it enters the protocol and replies via a message on name p , otherwise if it is in the busy state it communicates via name n instead. Hence, node α anticipates a message on exactly one of the names p and n and when one arrives both names are discarded.

Note that in the description above and hereafter, given an agent constant D we write $D(\tilde{u}_1, \dots, \tilde{u}_n)$ for $D(\tilde{u}_1 \psi \dots \psi \tilde{u}_n)$, where ψ is a (separator) name of a distinct sort Ψ which allows us to distinguish between the \tilde{u}_i 's.

The following definition allows us to capture the notion of connectivity within the system.

Definition 3.10.1 Let P be an agent of the form $\prod_{i \in I} P_i$ and S a sort. An S -path in P is a sequence $s_1, \dots, s_{n-1} : S$ such that for some $i_1, \dots, i_n \in I$, $s_j \in$

$\text{fn}(P_{i_j}) \cap \text{fn}(P_{i_{j+1}})$ for all $1 \leq j < n$. In such a case we say that *there is an S-path between P_{i_1} and P_{i_n}* .

It is convenient to introduce the following agents constants: $\text{Root} \equiv \text{Node}$ and $\text{Broot} \equiv \text{Busy}$. The system in its initial state, is represented by the agent

$$P_0 \stackrel{\text{def}}{=} (\nu \tilde{z})(\text{Root}(\text{start } \tilde{s}, v) \mid \prod_{i \in I} \text{Node}(\tilde{s}_i, v_i))$$

where the following are satisfied:

1. $\tilde{z} = \tilde{s} \cup \bigcup_{i \in I} \tilde{s}_i$,
2. for all $s \in \tilde{z}$, s is owned by exactly two components in P_0 , and
3. for all $i \in I$ there exists an L-path between $\text{Node}(\tilde{s}_i, v_i)$ and $\text{Root}(\text{start}, \tilde{s}, v)$.

Hence, the system consists of a number of nodes composed in parallel, where Root represents the root node containing the only free name of the system, start , (Clause 1) via which computation is triggered. Thus, in its initial state the system may receive a message along name start consisting of a pair (out, n) , where out is the name via which the outcome of the algorithm should be emitted to the environment. The second clause ensures that each link connects exactly two nodes. Finally, the last clause claims that there exists a path constructed of L links between each node and the root. This property captures the fact that the set of nodes is connected.

It is not difficult to see that a simpler description of the system is possible which replaces synchronizations via names of sort N by communications of integers via names of sort T, where the integers involved are discarded. One might even go a step further: a description of the system is possible which involves no names of sort T or N but where links of sort L serve a twofold purpose by being responsible both for activating nodes and for communicating integer values as required by the algorithm. This view of the system may be faithfully encoded in a non-mobile process calculus, such as value-passing CCS, and it would not be difficult to prove the CCS encoding equivalent to the π_v -calculus description. However, we believe that employing the π_v -calculus definition is advantageous for a number of reasons. First, by introducing mobility we obtain a dynamic description of the system which allows the extraction of information concerning the state of the system as computation proceeds by simply observing its structure. For example, the presence of names of sorts T and N in a derivative of the system, names which are created and transmitted during communications between the components, highlights the extent of the propagation. Moreover, by distinguishing between sorts T and N we make precise

the distinction between the interactions involved and thus provide an additional tool for formalizing intuitions concerning the correctness of the algorithm. As we will see in the the following section, this facilitates the reasoning regarding connectivity invariants satisfied by the system.

3.10.2 Correctness Proof

According to the specification, when the algorithm is invoked the system should compute the sum of the weights of its nodes and emit it to the environment. Hence the correctness property we want to establish is the following:

Theorem 3.10.2 $P_0 \approx \text{start}(\text{out}, n). \overline{\text{out}}\langle \text{sum} \rangle. 0$, where $\text{sum} = v + \sum_{i \in I} v_i$.

We will prove this result using the notions of coupled simulation and confluence. First, we will establish that there is a coupled simulation between P_0 and a system S_0 . Using the confluence of the latter system we will then show that

$$S_0 \approx \text{start}(\text{out}, n). \overline{\text{out}}\langle \text{sum} \rangle. 0.$$

In order to deduce the required result it remains to establish convergence of the systems involved. This result is also interesting in itself as it implies the absence of livelock.

A coupled simulation

As observed in [Vaa95] the algorithm may be thought of as having two phases. In the first phase, a spanning tree of the network of nodes is constructed with root the Root node. In the second phase, the spanning tree is used for computing the sum of all weights by ensuring that each node computes the sum of the weights of the nodes in the subtree rooted there and sending the value to its parent within the tree. When the execution of the algorithm begins, the creation of any spanning tree is possible and the choice of each of its edges from the set of links in the system takes place nondeterministically, step by step.

In order to restrict our attention to a simpler agent and reduce the nondeterminism exhibited by P_0 , we consider another system S_0 . This system consists of the summation of agents, each corresponding to a possible spanning tree of the network, and enforcing its creation. In this way, “nondeterminism is pushed backwards to the initial state” [Vaa95], and the number of possible actions at each step during computation is reduced. We will show that this simple system has the same observable behaviour as P_0 by establishing a coupled simulation between the two.

In fact, one might expect that a stronger relation holds: $P_0 \approx S_0$. However, a coupled simulation allows for a simpler treatment and is acceptable for our purposes – the fact that the systems are determinate allows us to conclude the stronger result. A similar approach is taken in the correctness proof of [Vaa95], where an automaton is designed with this intention, and a prophecy variable is established relating it to the original system.

We define agent Node' as follows:

$$\text{Node}'(s, \tilde{s}, v) \stackrel{\text{def}}{=} s(p, n). \text{Busy}(p, \tilde{s}, \tilde{s}, \varepsilon, \varepsilon, v)$$

Thus the agent Node' in its quiescent state may receive a message along a unique L-name s and then behave like agent Busy . Given agent P_0 defined in the previous section, we define S_0 as follows:

$$S_0 \stackrel{\text{def}}{=} \text{start}(\text{out}, n). \Sigma_{T \in ST(P_0)} \tau. T(\text{out})$$

where $T \in ST(P_0)$ iff

1. $T \equiv (\nu \tilde{x})(\text{Broot}(\text{out}, \tilde{s}, \tilde{s}, \varepsilon, \varepsilon, v) \mid \Pi_{i \in I} \text{Node}'(s_i, \tilde{s}_i - s_i, v_i))$ for $s_i \in \tilde{s}_i$ and
2. for all $i \in I$ there exists an L-path between $\text{Node}'(s_i, \tilde{s}_i, v_i)$ and Broot beginning with s_i .

Hence, if $T \in ST(P_0)$ then the agent T represents the same collection of nodes as P_0 connected by the same links with the distinction that each node is willing to receive its first message on exactly one of its links. The set of these distinguished links, $\{s_i\}_{i \in I}$, can be seen to form a spanning tree of the graph defined by the set of nodes: by construction (Clause 2) there exists a path between each node and the root composed entirely of links in $\{s_i\}_{i \in I}$ and $|\{s_i\}_{i \in I}| = n - 1$, where n is the number of nodes in the system. Thus, after a $\text{start}(\text{out}, n)$ action, S_0 may evolve into an agent T , where T represents one of the possible spanning trees of the system described by P_0 . By the construction it is easy to see that certain properties of P_0 are also satisfied by S_0 , namely,

3. $\text{fn}(S_0) = \{\text{start}\}$, and
4. for all $s \in \text{bn}(S_0)$, s is owned by exactly two components in S_0 .

Moreover, as we will see in detail later, T and P_0 have very similar behaviour: the transition system of T is a fragment of that of P_0 .

We now proceed to establish some invariant properties satisfied by P_0 and S_0 . The following lemma formalizes a number of properties satisfied by certain

derivatives of agent P_0 . An agent of the kind considered is a parallel composition of the root node in its busy state and a number of nodes in a busy or the quiescent state. It contains a single free name, out , and the remainder of its names are used in a disciplined way (Clause 3.10.3(3)). Further, the last two clauses ensure that the connectivity of the system is not lost during computation. In particular, there exists a T-path between each Busy component and the root, and an L-path between each Node component and either the root or a Busy node. Hence, nodes that have not yet entered the protocol are still connected by a path to either the root or a busy node and, in turn, busy nodes are connected to the root. We note that the T-paths represent the edges of the spanning tree as it is created.

To make the reasoning about the systems more readable, in the remainder of this section we will often write $\text{Busy}(\tilde{u})$ for $\text{Busy}(p, \tilde{s}, \tilde{s}', \tilde{x}, \tilde{y}, v)$ and similarly for Broot . We will also omit parameters where they are irrelevant to the discussion.

Lemma 3.10.3 Suppose $P_0 \xrightarrow{\text{start}(\text{out}, n)} P$. Then

$$P = (\nu \tilde{p})(\text{Broot}(\text{out}, \tilde{s}, \tilde{s}', \tilde{x}, \tilde{y}, v) \mid \prod_{j \in J} \text{Busy}(p_j, \tilde{s}_j, \tilde{s}_j', \tilde{x}_j, \tilde{y}_j, v_j) \mid \prod_{k \in K} \text{Node}(\tilde{s}_k, v_k))$$

where

1. $\text{fn}(P) = \{\text{out}\}$, and out is not borne in P ,
2. for all $s : L \in \text{bn}(P)$, s is owned by exactly two components in P and for all $r : T, N \in \text{bn}(P)$, r is uniquely borne and uniquely handled by at most one component of P ,
3. for all $j \in J$, there exists a T-path between $\text{Busy}(p_j)$ and Broot beginning with name p_j , and
4. for all $k \in K$, there exists an L-path between $\text{Node}(\tilde{s}_k, v_k)$ and either Broot or $\text{Busy}(\tilde{u}_j)$ for some $j \in J$.

PROOF: The proof is by induction on the length of the derivation. Clearly, the claim holds for the case $P_0 \xrightarrow{\text{start}(\text{out}, n)} P = (\nu \tilde{p})(\text{Broot}(\text{out}, \tilde{s}, \tilde{s}', \varepsilon, \langle p, n \rangle, v) \mid \text{Busy}(p, \tilde{s}_i, \tilde{s}_i', \varepsilon, \varepsilon, v_i) \mid \prod_{k \neq i} \text{Node}(\tilde{s}_k, v_k))$, by the construction of P_0 and since no names of sort T or N occur as yet in the system. So suppose $P_0 \xrightarrow{\text{start}(\text{out}, n)} P' \xrightarrow{\tau} P$, where

$$P' = (\nu \tilde{p})(\text{Broot}(\text{out}, \tilde{s}, \tilde{s}', \tilde{x}, \tilde{y}, v) \mid \prod_{j \in J} \text{Busy}(p_j, \tilde{s}_j, \tilde{s}_j', \tilde{x}_j, \tilde{y}_j, v_j) \mid \prod_{k \in K} \text{Node}(\tilde{s}_k, v_k))$$

and the claim holds for P' . Four possibilities exist for the transition $P' \xrightarrow{\tau} P$, one corresponding to the communication between a Node and a Busy component and

three corresponding to the possible communications between two Busy components (via names of sort L, T and N). We consider the two more interesting. The first corresponds to the communication between two Busy components via an L-name and the second to the activation of a Node by a Busy component.

- Suppose

$$\begin{array}{ll} \text{Busy}\langle p_l, \tilde{s}_l, \tilde{s}_l', \tilde{x}_l, \tilde{y}_l, v_l \rangle & \xrightarrow{(\nu q, n) \tilde{s}(q, n)} P_1 = \text{Busy}\langle p_l, \tilde{s}_l - s, \tilde{s}_l', \tilde{x}_l, \tilde{y}_l \langle q, n \rangle, v_l \rangle \\ \text{Busy}\langle p_k, \tilde{s}_k, \tilde{s}_k', \tilde{x}_k, \tilde{y}_k, v_k \rangle & \xrightarrow{s(q, n)} P_2 = \text{Busy}\langle p_k, \tilde{s}_k, \tilde{s}_k' - s, \tilde{x}_k n, \tilde{y}_k, v_k \rangle \end{array}$$

and

$$\begin{aligned} P = (\nu \tilde{p} q n) & (\text{Broot}\langle \text{out}, \tilde{s}, \tilde{s}', \tilde{x}, \tilde{y}, v \rangle \mid \Pi_{k \in K} \text{Node}\langle \tilde{s}_k, v_k \rangle \mid P_1 \mid P_2 \\ & \mid \Pi_{j \in J'} \text{Busy}\langle r_j, \tilde{s}_j, \tilde{s}_j', \tilde{x}_j, \tilde{y}_j, v_j \rangle), \end{aligned}$$

where $J' = J - \{k, l\}$. By construction, P is of the form required and clearly it satisfies the first two properties. Next we consider Property 3. It is clear that this transition has not affected any of the T-names of P' and $n(P)[T = n(P')[T \cup \{q\}]$. Hence, by the induction hypothesis the claim follows.

Finally, to verify the last property, we note that s is the only L-name affected by the transition. Moreover, by the induction hypothesis, s is owned by exactly two components in P' , namely the components engaging in the transition $P' \xrightarrow{\tau} P$. So suppose \tilde{s} is an L-path within P' between a Node and a Busy component and where only the last agent involved in the path is not in the quiescent state. Since s is owned by two Busy components, it does not occur in the L-path and so the same path occurs within P . Thus Property 4 is satisfied.

- Suppose

$$\begin{array}{ll} \text{Busy}\langle p_l, \tilde{s}_l, \tilde{s}_l', \tilde{x}_l, \tilde{y}_l, v_l \rangle & \xrightarrow{(\nu q, n) \tilde{s}(q, n)} P_1 = \text{Busy}\langle p_l, \tilde{s}_l - s, \tilde{s}_l', \tilde{x}_l, \tilde{y}_l \langle q, l \rangle, v_l \rangle, \\ \text{Node}\langle \tilde{s}_m, v_m \rangle & \xrightarrow{s(q, n)} P_2 = \text{Busy}\langle q, \tilde{s}_m - s, \tilde{s}_m - s, \varepsilon, \varepsilon, v_m \rangle \end{array}$$

and

$$\begin{aligned} P = (\nu \tilde{p} q n) & (\text{Broot}\langle \text{out}, \tilde{s}, \tilde{s}', \tilde{x}, \tilde{y}, v \rangle \mid \Pi_{k \in K'} \text{Node}\langle \tilde{s}_k, v_k \rangle \mid P_1 \mid P_2 \\ & \mid \Pi_{j \in J'} \text{Busy}\langle p_j, \tilde{s}_j, \tilde{s}_j', \tilde{x}_j, \tilde{y}_j, v_j \rangle), \end{aligned}$$

where $J' = J - \{l\}$ and $K' = K - \{m\}$. As before, P is of the form required and satisfies the first two properties. In addition, it is easy to see that, if \tilde{p} is a T-path from component P_1 to the root (which exists by the induction

hypothesis), then $q\tilde{p}$ is a T-path from P_2 to the root. Finally, since no L names have been affected by the transition other than s which was owned by the components $\text{Busy}\langle p_n, \widetilde{s_n}, \widetilde{s_n'}, \widetilde{x_n}, \widetilde{y_n}, v_n \rangle$ and $\text{Node}\langle \widetilde{s_m}, v_m \rangle$ of P' , it is easy to see that Property 4 is also satisfied.

This completes the proof. \square

We continue with a series of results that establish a correspondence between the execution paths of P_0 and S_0 . The next lemma formalizes the earlier intuition that any path of S_0 corresponding to some spanning tree can be matched by a path of P_0 constructing the same tree.

Lemma 3.10.4 Suppose

$$S_0 \xrightarrow{\text{start}\langle \text{out}, n \rangle} S = (\nu \tilde{p})(\text{Broot}\langle \text{out}, \tilde{u} \rangle \mid \Pi_{j \in J} \text{Busy}\langle \tilde{u}_j \rangle \mid \Pi_{k \in K} \text{Node}'\langle s_k, \widetilde{s_k}, v_k \rangle).$$

Then

$$P_0 \xrightarrow{\text{start}\langle \text{out}, n \rangle} P = (\nu \tilde{p})(\text{Broot}\langle \text{out}, \tilde{u} \rangle \mid \Pi_{j \in J} \text{Busy}\langle \tilde{u}_j \rangle \mid \Pi_{k \in K} \text{Node}\langle s_k, \widetilde{s_k}, v_k \rangle).$$

PROOF: The proof of this result is straightforward. It suffices to note that by the construction of S_0 , if $S_0 \xrightarrow{\alpha} S' \Rightarrow S$ then $\alpha = \text{start}\langle \text{out}, n \rangle$ and $S' \in ST(P_0)$. It is then easy to see that the transition system of S' is a fragment of that of P_0 and hence the result follows. \square

We have the following corollary.

Corollary 3.10.5 Suppose $S_0 \xrightarrow{\text{start}\langle \text{out}, n \rangle} S$. Then

$$S = (\nu \tilde{p})(\text{Broot}\langle \tilde{u} \rangle \mid \Pi_{j \in J} \text{Busy}\langle p_j, \tilde{s}_j, \tilde{s}_j', \tilde{x}_j, \tilde{y}_j, v_j \rangle \mid \Pi_{k \in K} \text{Node}'\langle s_k, \widetilde{s_k}, v_k \rangle)$$

where $J \cap K = \emptyset$ and the following hold:

1. $\text{fn}(S) = \{\text{out}\}$ and out is not borne in S ,
2. for all $s : L \in \text{bn}(S)$, s is owned by exactly two components in S and, for all $r : T, N \in \text{bn}(S)$, r is uniquely borne and at most uniquely handled in S ,
3. for all $j \in J$, there exists a T-path between $\text{Busy}\langle p_j, \tilde{s}_j, \tilde{s}_j', \tilde{x}_j, \tilde{y}_j, v_j \rangle$ and Broot beginning with name p_j , and
4. for all $k \in K$, there exists an L-path between $\text{Node}'\langle s_k, \widetilde{s_k}, v_k \rangle$ and either Broot or $\text{Busy}\langle p_j, \tilde{s}_j, \tilde{s}_j', \tilde{x}_j, \tilde{y}_j, v_j \rangle$ for some $j \in J$ beginning with name s_k .

PROOF: Suppose $S_0 \xrightarrow{\text{start}\langle \text{out}, n \rangle} S$ where S is as above. Then, by Lemma 3.10.4, $P_0 \xrightarrow{\text{start}\langle \text{out}, n \rangle} P$ where

$$P = (\nu \tilde{p})(\text{Broot}\langle \tilde{u} \rangle \mid \Pi_{j \in J} \text{Busy}\langle p_j, \tilde{s}_j, \tilde{s}_j', \tilde{x}_j, \tilde{y}_j, v_j \rangle \mid \Pi_{k \in K} \text{Node}\langle s_k, \tilde{s}_k, v_k \rangle)$$

Agent P satisfies the properties of Lemma 3.10.3. The similarity of the constructions of S and P and Clause (2) in the definition of $ST(P_0)$, allow us to conclude the result. \square

More interesting is the following lemma that claims that for any execution path of P_0 there exists an agent $T \in ST(P_0)$ which can mirror the transition.

Lemma 3.10.6 Suppose

$$P_0 \xrightarrow{\text{start}\langle \text{out}, n \rangle} P = (\nu \tilde{p})(\text{Broot}\langle \text{out}, \tilde{u} \rangle \mid \Pi_{j \in J} \text{Busy}\langle \tilde{u}_j \rangle \mid \Pi_{k \in K} \text{Node}\langle \tilde{s}_k, v_k \rangle).$$

Then there exists $T \in ST(P_0)$ such that

$$\begin{aligned} S_0 \xrightarrow{\text{start}\langle \text{out}, n \rangle} & \xrightarrow{\tau} T \equiv (\nu \tilde{z})(\text{Broot}\langle \text{out}, \tilde{s}, \tilde{s}, \varepsilon, \varepsilon, v \rangle \mid \Pi_{i \in I} \text{Node}'\langle s_i, \tilde{s}_i - s_i, v_i \rangle) \\ \Rightarrow & S = (\nu \tilde{p})(\text{Broot}\langle \text{out}, \tilde{u} \rangle \mid \Pi_{j \in J} \text{Busy}\langle \tilde{u}_j \rangle \mid \Pi_{k \in K} \text{Node}'\langle s_k, \tilde{s}_k - s_k, v_k \rangle). \end{aligned}$$

PROOF: The proof is by induction on the length of the derivation $P_0 \xrightarrow{\text{start}\langle \text{out}, n \rangle} P$. The base case is obvious. So suppose $P_0 \xrightarrow{\text{start}\langle \text{out}, n \rangle} P' \xrightarrow{\tau} P$ where

$$P' = (\nu \tilde{p})(\text{Broot}\langle \text{out}, \tilde{u} \rangle \mid \Pi_{j \in J} \text{Busy}\langle \tilde{u}_j \rangle \mid \Pi_{k \in K} \text{Node}\langle \tilde{s}_k, v_k \rangle)$$

and there exists $T \in ST(P_0)$ such that $S_0 \xrightarrow{\text{start}\langle \text{out}, n \rangle} \xrightarrow{\tau} T$ and

$$T \Rightarrow S' = (\nu \tilde{p})(\text{Broot}\langle \text{out}, \tilde{u} \rangle \mid \Pi_{j \in J} \text{Busy}\langle \tilde{u}_j \rangle \mid \Pi_{k \in K} \text{Node}\langle s_k, \tilde{s}_k - s_k, v_k \rangle).$$

The transition is the result either of a communication between two Busy components or a communication between a Busy and a Node component:

- Suppose

$$\begin{aligned} \text{Busy}\langle \tilde{u}_n \rangle & \xrightarrow{\alpha} P_1 \\ \text{Busy}\langle \tilde{u}_m \rangle & \xrightarrow{\bar{\alpha}} P_2 \end{aligned}$$

and

$$P = (\nu \tilde{p}\tilde{w})(\text{Broot}\langle \text{out}, \tilde{u} \rangle \mid P_1 \mid P_2 \mid \Pi_{j \in J'} \text{Busy}\langle \tilde{u}_j \rangle \mid \Pi_{k \in K} \text{Node}\langle \tilde{s}_k, v_k \rangle)$$

where $J' = J - \{m, n\}$, $\tilde{w} = \text{bn}(\alpha) \cup \text{bn}(\bar{\alpha})$, $P_1 = \text{Busy}\langle \tilde{u}_n' \rangle$ and either $\text{subj}(\alpha) : \top$ and $P_2 = 0$ or $P_2 = \text{Busy}\langle \tilde{u}_m' \rangle$. Since S' contains the same Busy-components as P' , it can mimic this transition to give $S' \xrightarrow{\tau} S$ where

$$S = (\nu \tilde{p}\tilde{w})(\text{Broot}\langle \text{out}, \tilde{u} \rangle \mid P_1 \mid P_2 \mid \Pi_{j \in J'} \text{Busy}\langle \tilde{u}_j \rangle \mid \Pi_{k \in K} \text{Node}'\langle s_k, \tilde{s}_k, v_k \rangle).$$

Since S has the form required this completes the case.

- Otherwise, suppose

$$\begin{array}{ll} \text{Busy}\langle p_l, \tilde{s}_l, \tilde{s}_l', \tilde{x}_l, \tilde{y}_l, v_l \rangle & \xrightarrow{(\nu q, n) \tilde{s}_i(p, n)} P_1 = \text{Busy}\langle p_l, \tilde{s}_l - s_i, \tilde{s}_l', \tilde{x}_l, \tilde{y}_l(p, l), v_l \rangle, \\ \text{Node}\langle \tilde{s}_m, v_m \rangle & \xrightarrow{s_i(p, n)} P_2 = \text{Busy}\langle p, \tilde{s}_m - s_i, \tilde{s}_m - s_i, \varepsilon, \varepsilon, v_m \rangle \end{array}$$

and

$$P = (\nu \tilde{p}pn)(\text{Broot}\langle \text{out}, \tilde{u} \rangle \mid P_1 \mid \Pi_{j \in J'} \text{Busy}\langle \tilde{u}_j \rangle \mid P_2 \mid \Pi_{k \in K'} \text{Node}\langle \tilde{s}_k, v_k \rangle)$$

where $J' = J - \{i\}$ and $K' = K - \{m\}$. Two possibilities exist. First suppose $s_i = s_m$. Then clearly S' can mimic this transition to give

$$\begin{aligned} S' \xrightarrow{\tau} S = (\nu \tilde{p}pn)(\text{Broot}\langle \text{out}, \tilde{u} \rangle \mid P_1 \mid \Pi_{j \in J'} \text{Busy}\langle \tilde{u}_j \rangle \mid P_2 \\ \mid \Pi_{k \in K'} \text{Node}'\langle s_k, \tilde{s}_k - s_k, v_k \rangle), \end{aligned}$$

which completes the case.

Otherwise, if $s_i \neq s_m$ then the transition can not be mirrored by S' . However, suppose $T = (\nu \tilde{s})(\text{Broot}\langle \text{out}, \tilde{s}, \tilde{s}, \varepsilon, \varepsilon, v \rangle \mid \Pi_{i \in I} \text{Node}'\langle s_i, \tilde{s}_i - s_i, v_i \rangle)$. Then we may construct T' as follows:

$$\begin{aligned} T' = (\nu \tilde{s})(\text{Broot}\langle \text{out}, \tilde{s}, \tilde{s}, \varepsilon, \varepsilon, v \rangle \mid \Pi_{i \neq m} \text{Node}'\langle s_i, \tilde{s}_i - s_i, v_i \rangle \\ \mid \text{Node}'\langle s_i, \tilde{s}_m - s_i, v_m \rangle). \end{aligned}$$

We claim that $T' \in ST(P_0)$. It suffices to show that there exists an L-path from $\text{Node}'\langle s_i, \tilde{s}_m - s_i, v_m \rangle$ to Broot beginning with s_i . In other words, if s_i is owned by $\text{Node}'\langle s_p, \tilde{s}_p, v_p \rangle$ for some $p \in I$, $p \neq m$, then there exists an L-path from this node to Broot. It is easy to see that, since the transition $T \Rightarrow S'$ does not involve $\text{Node}\langle s_i, \tilde{s}_i - s_i, v_i \rangle$, T' has the following transition.

$$\begin{aligned} T' \Rightarrow T_1 = (\nu \tilde{p})(\text{Broot}\langle \text{out}, \tilde{u} \rangle \mid \Pi_{j \in J'} \text{Busy}\langle \tilde{u}_j \rangle \mid \text{Busy}\langle \tilde{u} \rangle \\ \mid \text{Node}'\langle s_i, \tilde{s}_m - s_i, v_m \rangle \mid \Pi_{k \in K'} \text{Node}'\langle s_k, \tilde{s}_k - s_k, v_k \rangle) \end{aligned}$$

Since $\text{Busy}\langle p_l, \tilde{s}_l, \tilde{s}_l', \tilde{x}_l, \tilde{y}_l, v_l \rangle$ owns s_i it must be that for some $p \in I$ and some $s \in \text{Act}^*$,

$$\text{Node}\langle s_p, \tilde{s}_p, v_p \rangle \xRightarrow{s} \text{Busy}\langle s_l, \tilde{s}_l, \tilde{s}_l', \tilde{x}_l, \tilde{y}_l' \rangle.$$

It is easy to see by the construction of T that since this transition is possible, there exists an L-path from $\text{Node}\langle s_p, \tilde{s}_p, v_p \rangle$ to Root as required. Moreover,

$$\begin{aligned} T_1 \xrightarrow{\tau} T_2 = (\nu \tilde{p})(\text{Broot}\langle \text{out}, \tilde{u} \rangle \mid \Pi_{j \in J'} \text{Busy}\langle \tilde{u}_j \rangle \mid P_1 \mid P_2 \\ \mid \Pi_{k \in K'} \text{Node}'\langle s_k, \tilde{s}_k - s_k, v_k \rangle) \end{aligned}$$

and so $S_0 \xrightarrow{\tau} T' \Rightarrow T_2$ where T_2 is as required.

□

We may now define the relations which will be shown to comprise a coupled simulation between P_0 and S_0 .

Definition 3.10.7

- Let \mathcal{R}_1 be the smallest set such that

1. $(0, 0), (P_0, S_0) \in \mathcal{R}_1$,
2. if $P_0 \xrightarrow{\text{start}\langle \text{out}, n \rangle} P = (\nu \tilde{p})(\text{Broot}\langle \tilde{u} \rangle \mid \Pi_{j \in J} \text{Busy}\langle \tilde{u}_j \rangle \mid \Pi_{k \in K} \text{Node}\langle u_k \rangle)$ then, if $K \neq \emptyset$ then $(P, \Sigma_{T \in ST(P_0)} \tau.T\langle \text{out} \rangle) \in \mathcal{R}_1$, otherwise, if $K = \emptyset$, then $(P, P) \in \mathcal{R}_1$.

- Let \mathcal{R}_2 be the smallest set such that

1. $(0, 0), (P_0, S_0) \in \mathcal{R}_2$,
2. if $S_0 \xrightarrow{\text{start}\langle \text{out}, n \rangle} S = (\nu \tilde{p})(\text{Broot}\langle \tilde{u} \rangle \mid \Pi_{j \in J} \text{Busy}\langle \tilde{u}_j \rangle \mid \Pi_{k \in K} \text{Node}'\langle s_k, \tilde{s}_k, v_k \rangle)$ then $(P, S) \in \mathcal{R}_2$, where

$$P = (\nu \tilde{p})(\text{Broot}\langle \tilde{u} \rangle \mid \Pi_{j \in J} \text{Busy}\langle \tilde{u}_j \rangle \mid \Pi_{k \in K} \text{Node}\langle s_k \tilde{s}_k, v_k \rangle).$$

Thus, \mathcal{R}_1 relates P_0 with S_0 and all of its derivatives containing at least one Node component with $\Sigma_{T \in ST(P_0)} \tau.T\langle \text{out} \rangle$. It is reflexive for all other derivatives of P_0 . On the other hand, \mathcal{R}_2 relates P_0 with S_0 and derivatives of S_0 with appropriate agents containing Node as opposed to Node' agents. Our first major result is presented in the following lemma.

Lemma 3.10.8 $(\mathcal{R}_1, \mathcal{R}_2)$ is a coupled simulation.

PROOF: Suppose $(P, S) \in \mathcal{R}_1$ and $P \xrightarrow{\alpha} P'$. According to the definition either $P = P_0$ or $P_0 \xrightarrow{\text{start}\langle \text{out}, n \rangle} P$. In the former case, P is stable and if $P \xrightarrow{\alpha} P'$ then $\alpha = \text{start}\langle \text{out}, n \rangle$. Further, $S = S_0$ and by construction of \mathcal{R}_2 , $(P, S) \in \mathcal{R}_2$. Finally, $S \xrightarrow{\alpha} S' = \Sigma_{T \in ST(P_0)} \tau.T\langle \text{out} \rangle$ and by the construction of \mathcal{R}_1 , $(P', S') \in \mathcal{R}_1$ as required.

So consider the second case. The following possibilities exist.

1. $P = (\nu \tilde{p})(\text{Broot}\langle \tilde{u} \rangle \mid \Pi_{j \in J} \text{Busy}\langle \tilde{u}_j \rangle \mid \Pi_{k \in K} \text{Node}\langle \tilde{s}_k, v_k \rangle)$ where $K \neq \emptyset$. Then $S = \Sigma_{T \in ST(P_0)} \tau.T\langle \text{out} \rangle$. First, note that P is not stable. For, as suggested by Lemma 3.10.3(4), there must exist at least one internal action between a Node-component and a Busy-component. So consider the transition $P \xrightarrow{\alpha} P'$. The following possibilities exist:

- $\alpha = \tau$ and $P' = (\nu\tilde{p}')(\text{Broot}\langle\tilde{u}'\rangle \mid \Pi_{j \in J'} \text{Busy}\langle\tilde{u}_j'\rangle \mid \Pi_{k \in K'} \text{Node}'\langle\tilde{s}_k, v_k\rangle)$ where $K' \neq \emptyset$. Then, by definition, $(P', S) \in \mathcal{R}_1$ which completes the case.
- $\alpha = \tau$ and $P' = (\nu\tilde{p}')(\text{Broot}\langle\tilde{u}'\rangle \mid \Pi_{j \in J'} \text{Busy}\langle\tilde{u}_j'\rangle)$. By Lemma 3.10.4, $S \Rightarrow S' = (\nu\tilde{p})(\text{Broot}\langle\tilde{u}'\rangle \mid \Pi_{j \in J'} \text{Busy}\langle\tilde{u}_j'\rangle) = P'$. By definition of \mathcal{R}_1 , $(P', S') \in \mathcal{R}_1$ as required.

Note that there is no transition

$$\text{Broot}\langle\text{out}, \varepsilon, \varepsilon, \varepsilon, \varepsilon, v\rangle \xrightarrow{\alpha} 0, \text{ and}$$

$$P' = (\nu\tilde{p}')(\Pi_{j \in J'} \text{Busy}\langle\tilde{u}_j'\rangle \mid \Pi_{k \in K'} \text{Node}'\langle\tilde{s}_k, v_k\rangle)$$

with $\alpha = \overline{\text{out}}\langle v \rangle$ since then there exist no paths between component Broot and the remaining Busy and Node components of P' , contradicting Lemma 3.10.3.

2. $P = (\nu\tilde{p})(\text{Broot}\langle\tilde{u}\rangle \mid \Pi_{j \in J} \text{Busy}\langle\tilde{u}_j\rangle)$. By the construction of \mathcal{R}_1 , $S = P$. Moreover, by Lemma 3.10.4 since $P_0 \xrightarrow{\text{start}\langle\text{out}, n\rangle} P$, $S_0 \xrightarrow{\text{start}\langle\text{out}, n\rangle} S = P$ and by the construction of \mathcal{R}_2 , $(P, S) \in \mathcal{R}_2$. Hence, if $P \xrightarrow{\alpha} P'$ then $S \xrightarrow{\alpha} S' = P'$ and $(P', S') \in \mathcal{R}_1 \cap \mathcal{R}_2$ which completes the case.

The analysis of \mathcal{R}_2 uses similar arguments. □

Analysis of S_0

We begin by establishing that S_0 is a confluent agent. First we note that since each of Node , Node' and Root is capable of performing only finite computations before becoming inactive (of length $2n$ where n is the number of L-links owned by the component) we may deduce that P_0 and S_0 are convergent agents.

Lemma 3.10.9 S_0 is confluent.

PROOF: This result appeals to Lemma 3.5.6. The proof involves verifying that S_0 satisfies the conditions of the lemma.

First, we need to show that Node' and Root are \mathbf{L}_{TN} -confluent agents. So consider a derivative of Node' . This is of the form $\text{Busy}\langle p, \tilde{s}_1, \tilde{s}_2, \tilde{x}, \tilde{y}, v \rangle$. Let \mathcal{P} be the process system generated by Node' and let $\{\mathcal{P}^{\tilde{r}} \mid \tilde{r} \text{ a finite subset of } \mathbf{T} \times \mathbf{N}\}$ be the partition of \mathcal{P} defined by setting $\text{Busy}\langle p, \tilde{s}_1, \tilde{s}_2, \tilde{x}, \tilde{y}, v \rangle \in \mathcal{P}^{\tilde{y}}$. This partition is \mathbf{L}_{TN} -confluent as we can easily check that it is \mathbf{L}_{TN} -sensitive and the following holds:

if $Q \in \mathcal{P}^{\tilde{y}}$, that is $Q = \text{Busy}\langle p, \tilde{s}_1, \tilde{s}_2, \tilde{x}, \tilde{y}, v \rangle$ for some $p, \tilde{s}_1, \tilde{s}_2, \tilde{x}$ and v , and $Q \xrightarrow{\alpha} Q_1$, $Q \xRightarrow{\beta} Q_2$ where $(\text{subj}(\alpha), \text{subj}(\beta)) \notin \tilde{y}$, then by construction $Q_1 \xRightarrow{\beta} Q'$ and $Q_2 \xRightarrow{\alpha} Q'$.

Moreover, by Clause 3 in the definition of S_0 , $\text{fn}(S_0) = \{\text{start}\}$ where start is uniquely borne and not handled in S_0 by the Root-component and all of the remaining names of S_0 are owned by exactly two components. Further, by Corollary 3.10.5(1), for all derivatives S of S_0 , if $x \in \text{fn}(S)$ then x is uniquely handled and not borne in S , and if $x \in \text{bn}(S)$ then x is owned by at most two components in S . Finally, it is easy to see that Node' and Root are consistent agents. Hence by Lemma 3.5.6, S_0 is confluent. \square

Since S_0 is confluent, if $T \in ST(P_0)$ then T , being a derivative of S_0 , is confluent. In the next lemma we make use of the confluence of T and, by studying a fragment of its transition system, we conclude the following:

Lemma 3.10.10 Let $T \in ST(P_0)$. Then $T \approx \overline{\text{out}}(\text{sum}).0$.

PROOF: We will show that $T \Rightarrow \overline{\text{out}}(\text{sum}).0$. Being a derivative of S_0 , T is confluent and so by Lemma 3.4.10, $T \approx \overline{\text{out}}(\text{sum}).0$ as required.

So consider

$$T = (\nu \tilde{s})(\text{Broot}(\text{out}, \tilde{s}, \tilde{s}, \varepsilon, \varepsilon, v) \mid \prod_{i \in I} \text{Node}'(s_i, \tilde{s}_i, v_i)).$$

By Lemma 3.10.5(4), T may engage in the following transition:

$$\begin{aligned} T &\xrightarrow{\tau} (\nu \tilde{z}, p_k, n_k)(\text{Broot}(\text{out}, \tilde{s} - s_k, \tilde{s}, \varepsilon, \langle p_k, n_k \rangle, v) \mid \prod_{i \neq k} \text{Node}'(s_i, \tilde{s}_i, v_i) \\ &\quad \mid \text{Busy}(p_k, \tilde{s}_k - s_k, \tilde{s}_k - s_k, \varepsilon, \varepsilon, v_k)) \\ &\Rightarrow \dots \\ &\xrightarrow{\tau} T_1 = (\nu \tilde{z}')(\text{Broot}(\text{out}, \tilde{p}, \tilde{p}', \varepsilon, \tilde{y}, v) \mid \prod_{i \in I} \text{Busy}(p_i, \tilde{p}_i, \tilde{p}_i', \varepsilon, \tilde{y}_i, v_i)) \end{aligned}$$

This transition corresponds to the creation of the spanning tree which results in all of the nodes entering the protocol. It is then easy to see using Lemma 3.10.5(2) that T_1 may now behave as follows:

$$\begin{aligned} T_1 &\Rightarrow T_2 = (\nu \tilde{s}')(\text{Broot}(\text{out}, \varepsilon, \varepsilon, \tilde{x}, \tilde{y}, v) \mid \prod_{i \in I} \text{Busy}(p_i, \varepsilon, \varepsilon, \tilde{x}_i, \tilde{y}_i, v_i)) \\ &\Rightarrow T_3 = (\nu \tilde{s}')(\text{Broot}(\text{out}, \varepsilon, \varepsilon, \varepsilon, \tilde{y}', v) \mid \prod_{i \in I} \text{Busy}(p_i, \varepsilon, \varepsilon, \varepsilon, \tilde{y}_i', v_i)) \end{aligned}$$

The first transition corresponds to all nodes communicating with their non-parent neighbours via the links of sort L and subsequently the second transition captures the signaling along all N names handled in T_2 . Note that up to this point the weight of each node has not changed.

According to Property 2 of Lemma 3.10.5, all names of sort T except out are uniquely borne and at most uniquely handled in T_2 . Since a component

$$\text{Busy}(p_i, \varepsilon, \varepsilon, \varepsilon, \tilde{y}_i, v)$$

handles name p_i and bears names \tilde{y}_i , for all i there exists j such that $(p, n) \in \tilde{y}_j$ for some n . Hence

$$\sum_{i \in I} |\tilde{y}_i| + |\tilde{y}| \geq |\{p_i\}_{i \in I}| = |I|,$$

where \tilde{y}_i, \tilde{y} are the T -names borne in T_3 and $\{p_i\}_{i \in I}$ are the names handled in T_3 . Note that assuming $I \neq \emptyset$, by Lemma 3.10.5(3), $|\tilde{y}| \neq 0$. Hence there exists at least one $j \in I$ such that $|\tilde{y}_j| = 0$ and $\text{Busy}\langle p_j, \varepsilon, \varepsilon, \varepsilon, \varepsilon, v_j \rangle \xrightarrow{\overline{p_j}(v_j)} 0$. So

$$\begin{aligned} T_3 \xrightarrow{\tau} T_4 = (\nu \tilde{s}')(&\text{Broot}\langle \text{out}, \varepsilon, \varepsilon, \varepsilon, \tilde{y}', v \rangle \mid \Pi_{i \in I'} \text{Busy}\langle p_i, \varepsilon, \varepsilon, \varepsilon, \tilde{y}_i', v_i \rangle \\ &\mid \text{Busy}\langle p_k, \varepsilon, \varepsilon, \varepsilon, \tilde{y}_k - \langle p_j, n_j \rangle, v_k + v_j \rangle) \end{aligned}$$

where $I' = I - \{j, k\}$. Note that the sum of weights of nodes in the tree remains constant. The same argument can be used to explain the following transition:

$$T_4 \implies T_5 = \text{Broot}\langle \text{out}, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \text{sum} \rangle$$

and $T_5 \xrightarrow{\overline{\text{out}}(\text{sum})} 0$ as required. \square

Returning to S_0 it is easy to see by construction that if $S_0 \xrightarrow{\alpha} S$ then $\alpha = \text{start}\langle \text{out}, n \rangle$ and $S \in ST(P_0)$. Since by the previous lemma $S \approx \overline{\text{out}}\langle \text{sum} \rangle$, we conclude that $S_0 \approx \text{start}\langle \text{out}, n \rangle. \overline{\text{out}}\langle \text{sum} \rangle$.

Since P_0 and S_0 are convergent agents, by transitivity of $=_{cs}$ for convergent systems and the fact that $S_0 =_{cs} P_0$,

$$P_0 =_{cs} \text{start}\langle \text{out}, n \rangle. \overline{\text{out}}\langle \text{sum} \rangle. 0.$$

Since $\text{start}\langle \text{out}, n \rangle. \overline{\text{out}}\langle \text{sum} \rangle. 0$ is a determinate agent, by Propositions 3.3.7 and 2.4.6, P_0 is also determinate. Further, as $=_{cs} \subset \approx_{tr}$, by Lemma 3.1.7,

$$P_0 \approx \text{start}\langle \text{out}, n \rangle. \overline{\text{out}}\langle \text{sum} \rangle. 0.$$

This completes the correctness proof. \square

Chapter 4

Partial Confluence

In Chapter 3 we introduced some notions of determinacy and confluence in the context of the π_v -calculus and investigated their theory. In particular, we observed that confluent systems satisfy some interesting properties which may prove useful for reasoning about them.

However, it is often the case that systems have some *partial confluence* properties without being confluent. Thus a question arising is which such properties can be captured formally and exploited for process verification. The study of notions of partial confluence was initiated in [Tof90] where the intention was to identify conditions under which combinations of non-confluent agents yield confluent systems. A notion of *semi confluence* was introduced and its basic theory was developed. Another notion was defined in that work, due to Milner: *K-partial confluence*, where K is a set of visible labels. A K -partial confluent agent is required to be confluent with respect to K -actions, but no restrictions are imposed on the occurrences of the remaining actions. No results were proved in [Tof90] about K -partial confluence though the expectation was stated that it is preserved by a set of constructions. Another notion of partial confluence was introduced in [LW95a]. Though related to K -partial confluence, this notion, referred to as *R-confluence*, differs significantly from K -partial confluence in that it does not require agents to be semantically invariant under silent actions. The motivation behind its definition was to study systems composed of non-confluent but well-behaved components (capable of state-changing τ -actions) whose interactions are of a certain disciplined kind. The theory of *R-confluence* was developed in the setting of CCS and extended to a mobile process calculus.

The definitions of the notions of partial confluence mentioned above involve bisimilarities, weak bisimilarity in [Tof90] and branching bisimilarity in [LW95a].

The first part of this chapter is concerned with extending the partial confluence theory of [LW95a] to take explicit account of divergence. This extension turns out to be quite complicated. It is based on the divergence-sensitive variant of branching bisimilarity, db-bisimilarity, and it is carried out in the context of the π_v -calculus. The motivation behind the development will become apparent in Chapter 6 where the theory is applied to prove the correctness of transformation rules for concurrent-object programs. Briefly, it is necessary because in that context divergence is considered to be an undesirable property of the systems studied and thus it is appropriate to employ a notion of observation equivalence that takes account of infinite silent paths. The theory also extends the results of [LW95a] in that it considers a polymorphic sorting discipline. In addition, this chapter presents a variant of db-bisimilarity which imposes requirements on convergence only with respect to a certain type of actions: it is possible for two systems, exactly one of which is divergent, to be related to each other, provided that the convergent agent may become divergent by performing a further action of the type specified. The main result concerning this relation is that the differences in divergent behaviour exhibited by two related systems are lost within certain contexts.

One of the most significant achievements of the partial confluence theory of [LW95a] is a result stating that when reasoning about certain systems, it is sufficient to examine in detail only a part of their behaviour. The systems considered could be viewed as consisting of two agents, P and T , which interact in a question-answer fashion, with possibly many questions outstanding at any moment. An important property is that on accepting a question from P , T immediately assumes a state in which the answer to that question is determined. Moreover, it restricts attention to the setting where when a question is invoked in T , the name which is supplied for return of an answer is distinct from names supplied in other question invocations for returning answers. The second part of the chapter presents an extension of the theory by relaxing these two assumptions. First, it considers systems where the processing of a question by T may result in a change of the state. Secondly, it considers systems where the assumption concerning the uniqueness of return links is dropped. The exposition does not take divergence into account, which is appropriate for our purposes. However, the results may be extended to do so along lines similar to those of Section 4.1. Note that since value-passing CCS is a subset of the π_v -calculus (where no name-passing is involved), the results we present in this chapter also hold in that setting.

4.1 Partial confluence and divergence

In [LW95a], a notion of partial confluence was introduced which proved useful in reasoning about classes of systems in which a subsystem interacts in a disciplined manner with a possibly non-confluent environment. Its basic theory was developed and it was used to prove the indistinguishability of two symbol-table classes, expressed in a concurrent-object programming language, in an arbitrary program context. Here we present an extension of that theory to accommodate divergence and take account of the polymorphism of the sorting discipline. The generalization turns out to be rather complicated: the definitions are more complex and the proofs are more involved as we need to take account of how divergence may appear in a system.

We begin by recalling the definition and some basic results of partial confluence as presented in [LW95a], to which we refer for the proofs. First some notation.

Notation 4.1.1 $P \Rightarrow \xrightarrow{\alpha} P'$ means that $P \Rightarrow P'' \xrightarrow{\alpha} P'$ for some P'' with $P'' \simeq P$, and moreover if $\alpha = \tau$ then $P' \not\approx P$.

Definition 4.1.2 Let R be a sort. A process P is R -confluent if for every derivative Q of P :

1. if $\rho \in R^\pm$, $\text{subj}(\alpha) \neq \text{subj}(\rho)$, $Q \xrightarrow{\rho} Q_1$ and $Q \Rightarrow \xrightarrow{\alpha} Q_2$ then for some Q' , $Q_1 \Rightarrow \xrightarrow{\alpha} Q'$ and $Q_2 \Rightarrow \xrightarrow{\rho} \simeq Q'$;
2. if $\rho_1, \rho_2 \in R^-$, $\text{subj}(\rho_1) = \text{subj}(\rho_2)$, $Q \xrightarrow{\rho_1} Q_1$ and $Q \Rightarrow \xrightarrow{\rho_2} Q_2$ then $\rho_1 = \rho_2$ and $Q_1 \simeq Q_2$;
3. if $\rho \in R^+$, $Q \xrightarrow{\rho} Q_1$ and $Q \Rightarrow \xrightarrow{\rho} Q_2$ then $Q_1 \simeq Q_2$.

R -confluence (unlike K -partial confluence) does not imply τ -inertness, and may thus be used for reasoning about non- τ -inert agents, as was the intention in [LW95a]. Essential to achieve this property is the use of branching bisimilarity: replacing \simeq in the definition by \approx is not satisfactory since the resulting notion is not preserved by bisimilarity. For further discussion see [LW95a]. However, the notion above is well-behaved, as the following holds:

Lemma 4.1.3 If P is R -confluent and $P \simeq Q$ then Q is R -confluent. □

The following notation is useful.

Notation 4.1.4 For $s = \alpha_1 \dots \alpha_n \in \text{Act}^*$ and a relation \asymp , we write \xrightarrow{s}_\asymp for the composite relation $\Rightarrow \xrightarrow{\alpha_1} \dots \Rightarrow \xrightarrow{\alpha_n} \asymp$. Moreover, we write $P \xrightarrow{\tau}_\asymp P'$, if

$P \Rightarrow P'' \xrightarrow{\tau} \asymp P'$, where $P \asymp P''$ and $P' \not\asymp P$, and we refer to $\xrightarrow{\tau} \asymp$ as a *decisive* (silent) action. \square

The following result states a useful property of R -confluent agents.

Lemma 4.1.5 Suppose P is R -confluent, $s \in R^{\pm*}$, $P \xrightarrow{s} \simeq P_1$ and $P \Rightarrow \xrightarrow{\alpha} P_2$, where if $\alpha \in R^+$ and $s = s_0 \rho s_1$, $\text{subj}(\rho) = \text{subj}(\alpha)$ and $\text{subj}(\alpha) \not\subseteq \text{subj}(s_0)$, then $\alpha = \rho$. Then either $\alpha \not\subseteq s$ and for some P_0 , $P_1 \Rightarrow \xrightarrow{\alpha} P_0$ and $P_2 \xrightarrow{s} \simeq P_0$, or $\alpha \in s$ and $P_2 \xrightarrow{s/\alpha} \simeq P_1$. \square

Moreover, the \simeq -state of a restricted composition of two R -confluent agents is not altered up to \simeq by an interaction between the components via a name in R .

Lemma 4.1.6 Suppose

1. P and T are R -confluent;
2. $P \xrightarrow{\rho} \simeq P'$, $T \xrightarrow{\bar{\rho}} \simeq T'$, where $\rho = r\langle\tilde{v}\rangle$ $\bar{\rho} = (\nu\tilde{u})\bar{r}\langle\tilde{v}\rangle$, $r : R$;
3. $(\nu\tilde{z})(P \mid T)$ is R -closed.

Then $(\nu\tilde{z})(P \mid T) \simeq (\nu\tilde{z}\tilde{u})(P' \mid T')$. \square

We now proceed to extend this theory to take account of divergence. To achieve this we work with \simeq_1 , the divergence-sensitive version of branching bisimilarity. The appropriate notion of partial confluence, R^1 -confluence, follows. Note that since we are now concerned with \simeq_1 as opposed to \simeq , for the remainder of this section $P \Rightarrow \xrightarrow{\alpha} P'$ means that $P \Rightarrow P'' \xrightarrow{\alpha} P'$ for some P'' with $P'' \simeq_1 P$, and moreover if $\alpha = \tau$ then $P' \not\asymp_1 P$.

Definition 4.1.7 Let R be a sort. A process P is R^1 -confluent if for every $Q \in \text{cc}(P)$:

1. if $\rho \in R^{\pm}$, $\text{subj}(\alpha) \neq \text{subj}(\rho)$, $Q \downarrow \rho\alpha$, $Q \xrightarrow{\rho} Q_1$ and $Q \Rightarrow \xrightarrow{\alpha} Q_2$, then $Q \downarrow \alpha\rho$ and for some Q' , $Q_1 \Rightarrow \xrightarrow{\alpha} Q'$ and $Q_2 \Rightarrow \xrightarrow{\rho} \simeq_1 Q'$;
2. if $\rho_1, \rho_2 \in R^-$, $\text{subj}(\rho_1) = \text{subj}(\rho_2)$, $Q \downarrow \rho_1$, $Q \downarrow \rho_2$, $Q \xrightarrow{\rho_1} Q_1$ and $Q \Rightarrow \xrightarrow{\rho_2} Q_2$, then $\rho_1 = \rho_2$ and $Q_1 \simeq_1 Q_2$;
3. if $\rho \in R^+$, $Q \downarrow \rho$, $Q \xrightarrow{\rho} Q_1$ and $Q \Rightarrow \xrightarrow{\rho} Q_2$, then $Q_1 \simeq_1 Q_2$.

It is required of an R^1 -confluent agent that each agent in its convergent core satisfies the following: (1) it enjoys a confluence property with respect to R -actions and arbitrary actions which do not together introduce divergence; (2) an output

action with a subject in R has an object that is uniquely determined (recall that $\rho_1 = \rho_2$ means that the actions are alpha-convertible) and leads to a uniquely determined state, assuming it does not cause divergence; and (3) an input action with a subject in R leads to a uniquely determined state, assuming again that it does not introduce divergence. It is easy to see that a weaker notion of R^l -confluence (and R -confluence) can be defined in the vein of Definition 3.4.3, by considering, in Clause (1), completion of the confluence diagram with respect to the weights of the actions involved, i.e. $\alpha[\rho]$ and $\rho[\alpha]$. To keep the presentation as simple as possible, we opt for the stronger notion. As a result, a property of an R^l -confluent agent is that may not transmit the same fresh name via two distinct subjects. Nonetheless, extending the results for the weaker notion is only a matter of an appropriate rewrite of the proofs.

The following properties are satisfied by an R^l -confluent agent:

Lemma 4.1.8 Suppose P is R^l -confluent, $\rho \in R^\pm$, $P \downarrow \rho$ and $P \Rightarrow \xrightarrow{\rho} P_1$. Then the following hold:

1. If $P \Rightarrow \xrightarrow{\tau} P_2$ then there exist P'_1, P'_2 such that $P_1 \Rightarrow \xrightarrow{\tau} P'_1$ and $P_2 \Rightarrow \xrightarrow{\rho} P'_2 \simeq_1 P'_1$.
2. If $P \Rightarrow P_2$ then there exist P'_1, P'_2 such that $P_1 \Rightarrow P'_1$ and $P_2 \Rightarrow \xrightarrow{\rho} P'_2 \simeq_1 P'_1$.
3. If $P \Rightarrow \xrightarrow{\rho} P_2$ then $P_1 \simeq_1 P_2$.
4. If $P \xrightarrow{\rho} P_2$ then there exists P'_2 such that $P_1 \Rightarrow P'_2$ with $P_2 \simeq_1 P'_2$. □

The following lemma shows that db-bisimilarity preserves R^l -confluence.

Lemma 4.1.9 If P is R^l -confluent and $P \simeq_1 Q$ then Q is R^l -confluent.

PROOF: Let $Q_0 \in \text{cc}(Q)$. Then since $P \simeq_1 Q$ there is $P_0 \in \text{cc}(P)$ such that $P_0 \simeq_1 Q_0$.

1. Suppose $Q_0 \downarrow \rho\alpha$, $Q_0 \xrightarrow{\rho} Q_1$ and $Q_0 \Rightarrow \xrightarrow{\alpha} Q_2$, where $\rho \in R^\pm$ and $\text{subj}(\alpha) \neq \text{subj}(\rho)$. Since $Q_0 \simeq_1 P_0$, $P_0 \downarrow \rho\alpha$ and $P_0 \Rightarrow P'_0 \xrightarrow{\rho} P_1$ for some P'_0, P_1 with $P'_0 \simeq_1 P_0$, $P_1 \simeq_1 Q_1$. Suppose $Q_0 \uparrow \alpha$. Then $\alpha \neq \tau$, $P_0 \uparrow \alpha$, $P'_0 \uparrow \alpha$ and since $P'_0 \downarrow$, $P'_0 \Rightarrow P_m \xrightarrow{\alpha} \uparrow$ for some P_m such that if $P_m \xrightarrow{\tau} P'_m$ then $P'_m \downarrow \alpha$ (note that such a state must exist since $P'_0 \downarrow$). By Lemma 4.1.8(2), $P_m \Rightarrow P''_m \xrightarrow{\rho} P'_m$ and $P_1 \Rightarrow \simeq_1 P'_m$, where $P_m \simeq_1 P''_m$. By the construction of P_m it must be that $P_m \equiv P''_m$ and thus $P_m \xrightarrow{\rho} P'_m$. Moreover, $P_m \downarrow \rho\alpha$, as $P_0 \downarrow \rho\alpha$, and so by R^l -confluence, $P_m \downarrow \alpha\rho$. This is a contradiction to $P_m \uparrow \alpha$. So it must be that $Q_0 \downarrow \alpha$. Thus, since $P'_0 \simeq_1 Q_0$ and $Q_0 \downarrow \alpha$ by \simeq_1 , $P'_0 \Rightarrow \xrightarrow{\alpha} P_2 \simeq_1 Q_2$.

Moreover, as $P'_0 \in \text{cc}(P'_0)$, P'_0 is R^l -confluent, so $P_0 \downarrow \alpha\rho$ and there exist P'_1, P'_2 , such that $P_1 \Rightarrow \xrightarrow{\alpha} P'_1$ and $P_2 \Rightarrow \xrightarrow{\rho} P'_2 \simeq P'_1$. Hence $Q_0 \downarrow \alpha\rho$ and since $P_1 \simeq_1 Q_1, P_2 \simeq_1 Q_2, Q_1 \Rightarrow \xrightarrow{\alpha} Q'_1 \simeq_1 P'_1, Q_2 \Rightarrow \xrightarrow{\rho} Q'_2 \simeq_1 P'_2$, where $Q'_1 \simeq_1 Q'_2$ as required.

2. Suppose $Q_0 \xrightarrow{\rho_1} Q_1$ and $Q_0 \Rightarrow Q'_2 \xrightarrow{\rho_2} Q_2$, where $\rho_1, \rho_2 \in R^-, \text{subj}(\rho_1) = \text{subj}(\rho_2)$ and $Q_0 \downarrow \rho_1, Q_0 \downarrow \rho_2$. Then $P_0 \downarrow \rho_1, P_0 \downarrow \rho_2$ and by R^l -confluence $P_0 \Rightarrow P'_0 \xrightarrow{\rho_1} P_1$ with $P'_0 \simeq_1 P_0$ and $P_1 \simeq_1 Q_1$. Moreover, as $P'_0 \simeq_1 Q_0, P'_0 \Rightarrow \xrightarrow{\rho_2} P_2$ with $P_2 \simeq_1 Q_2$. Since P'_0 is R^l -confluent, $\rho_1 = \rho_2$ and $P_1 \simeq_1 P_2$. Hence $Q_1 \simeq_1 Q_2$.
3. Suppose $Q_0 \xrightarrow{\rho} Q_1$ and $Q_0 \Rightarrow Q'_2 \xrightarrow{\rho} Q_2$, where $\rho \in R^+$ and $Q_0 \downarrow \rho$. Then $P_0 \downarrow \rho$ and $P_0 \Rightarrow P'_0 \xrightarrow{\rho} P_1$ with $P_0 \simeq_1 P'_0$ and $P_1 \simeq_1 Q_1$. As $P'_0 \simeq_1 Q_0, P'_0 \Rightarrow \xrightarrow{\rho} P_2$ with $P_2 \simeq_1 Q_2$. Since P'_0 is R^l -confluent, $P_1 \simeq_1 P_2$ and so $Q_1 \simeq_1 Q_2$. \square

Using the previous lemma we extend the properties of Lemma 4.1.8 for transitions of the form $\xrightarrow{\cdot} \simeq_1$ containing sequences of actions.

Lemma 4.1.10 Suppose P is R^l -confluent, $s \in R^{\pm*}, P \downarrow s$ and $P \xrightarrow{s} \simeq_1 P_1$. Then the following hold:

1. If $P \Rightarrow \xrightarrow{\tau} P_2$ then there exists P' such that $P_1 \Rightarrow \xrightarrow{\tau} P'$ and $P_2 \xrightarrow{s} \simeq_1 P'$.
2. If $P \Rightarrow P_2$ then there exists P' such that $P_1 \Rightarrow P'$ and $P_2 \xrightarrow{s} \simeq_1 P'$.
3. If $P \xrightarrow{s} \simeq_1 P_2$ then $P_1 \simeq_1 P_2$.
4. If $P \xRightarrow{s} P_2$ then $P_1 \Rightarrow P'_2$ with $P_2 \simeq_1 P'_2$. \square

The proofs of the properties follow easily from the definition of R^l -confluence. They are useful in proving the following result which extends the confluence properties satisfied by R^l -confluent agents to derivations under sequences of R -actions.

Lemma 4.1.11 Suppose P is R -confluent, $s \in R^{\pm*}, P \xrightarrow{s} \simeq_1 P_1$ and $\alpha \in \text{Act}$, where if $\alpha \in R^-$ and $s = s_0 \rho s_1$ where $\text{subj}(\rho) = \text{subj}(\alpha)$ and $\text{subj}(\alpha) \notin \text{subj}(s_0)$, then $\alpha = \rho$.

1. If $P \downarrow s$ and $P_1 \downarrow \alpha$, then $P \downarrow s\alpha$.
2. If $P \downarrow s\alpha$ then $P \downarrow t\alpha$ for any prefix t of s .
3. If $P \downarrow s\alpha, P \Rightarrow \xrightarrow{\alpha} P_2$ and $\alpha \notin s$, then $P \downarrow \alpha s$ and for some $P_0, P_1 \Rightarrow \xrightarrow{\alpha} P_0$ and $P_2 \xrightarrow{s} \simeq_1 P_0$.

4. If $P \downarrow s$, $\alpha \in s$ and $P \Rightarrow \xrightarrow{\alpha} P_2$ then $P_2 \xrightarrow{s/\alpha} \simeq_1 P_1$.

PROOF: Suppose $P \downarrow s$ and suppose $P \xRightarrow{s} P_2$. Then by Lemma 4.1.10(4), $P_1 \Rightarrow \simeq_1 P_2$. As $P_1 \downarrow \alpha$ then clearly $P_2 \downarrow \alpha$, and since P_2 is an arbitrary s -derivative of P , $P \downarrow s\alpha$.

The proof of Property 2 is by induction on s . If $s = \varepsilon$ then the claim follows immediately. So suppose $P \downarrow s\alpha$, where $s = s_0\rho$, $s_0 \in R^{\pm*}$ and $\rho \in R^{\pm}$. By the induction hypothesis, if $P \downarrow s_0\alpha$ then $P \downarrow t\alpha$ for any prefix t of s_0 . So, it is sufficient to prove that if $P \downarrow s\alpha$ then $P \downarrow s_0\alpha$. Suppose $P \downarrow s\alpha$. Certainly $P \downarrow s_0$ so suppose $P \xrightarrow{s_0} \simeq_1 P'_1 \xrightarrow{\rho} \simeq_1 P_1$. Then $P'_1 \downarrow \rho\alpha$. Suppose $P'_1 \uparrow \alpha$. Since $P'_1 \downarrow$, it must be that $P'_1 \Rightarrow P_m \xrightarrow{\alpha} \uparrow$ for some P_m such that if $P_m \xrightarrow{\tau} P'_m$ then $P_m \downarrow \alpha$. By Lemma 4.1.8(2), $P_m \Rightarrow P''_m \xrightarrow{\rho} P'_m$ and $P_1 \Rightarrow P''_1 \simeq_1 P'_m$, where $P_m \simeq_1 P''_m$. By the construction of P_m it must be that $P_m \equiv P''_m$ and thus $P_m \xrightarrow{\rho} P'_m$. Moreover, $P_m \downarrow \rho\alpha$ and so by R^1 -confluence, $P_m \downarrow \alpha\rho$. This is a contradiction to $P_m \uparrow \alpha$. So $P'_1 \downarrow \alpha$ and by part 1, $P \downarrow s_0\alpha$ as required.

The proof of Property 3 is by induction on s . If $s = \varepsilon$ then the fact follows immediately from the definition of \simeq_1 . So suppose $s = s_0\rho$ where $s_0 \in R^{\pm*}$, $\rho \in R^{\pm}$ and $P \xrightarrow{s_0} \simeq_1 P'_1 \xrightarrow{\rho} \simeq_1 P_1$. By the previous part we know that $P \downarrow s_0\alpha$. By the induction hypothesis there exists P'_0 such that $P'_1 \Rightarrow \xrightarrow{\alpha} P'_0$ and $P_2 \xrightarrow{s_0} \simeq_1 P'_0$. By Lemma 4.1.9, P'_1 is R^1 -confluent and by the assumption of the lemma, since $\alpha \neq \rho$, $\text{subj}(\alpha) \neq \text{subj}(\rho)$. So as $P'_1 \downarrow \rho\alpha$, we have that $P'_1 \downarrow \alpha\rho$ and P_0 can be found such that $P_1 \Rightarrow \xrightarrow{\alpha} P_0$ and $P'_0 \xrightarrow{\rho} \simeq_1 P_0$ as illustrated:

$$\begin{array}{ccccc} P & \xrightarrow{s_0} \simeq_1 & P'_1 & \xrightarrow{\rho} \simeq_1 & P_1 \\ \downarrow \alpha & & \downarrow \alpha & & \downarrow \alpha \\ P_2 & \xrightarrow{s_0} \simeq_1 & P'_0 & \xrightarrow{\rho} \simeq_1 & P_0 \end{array}$$

Finally, the proof of Property 4 is by induction on s . Suppose $s = s_0\rho$ where $s_0 \in R^{\pm*}$ and $\rho \in R^{\pm}$ and suppose $P \xrightarrow{s_0} \simeq_1 P'_1 \xrightarrow{\rho} \simeq_1 P_1$. Since $P \downarrow s_0$, if $\alpha \in s_0$ then by the induction hypothesis $P_2 \xrightarrow{s_0/\alpha} \simeq_1 P'_1$. Since $s/\alpha = (s_0/\alpha)\rho$, $P_2 \xrightarrow{s/\alpha} \simeq_1 P_1$. If $\alpha \notin s_0$ then by the previous part there exists P'_0 such that $P'_1 \Rightarrow \xrightarrow{\alpha} P'_0$ and $P_2 \xrightarrow{s_0} \simeq_1 P'_0$. Moreover, it must be that $\alpha = \rho$ and by Lemma 4.1.10(3), $P'_0 \simeq_1 P_1$, so again $P_2 \xrightarrow{s/\alpha} \simeq_1 P_1$. \square

As we have seen in the case of d-confluence, care is required in handling the possibility that a divergent agent may evolve to a convergent state. This can be handled satisfactorily by employing the following definition.

Definition 4.1.12 An agent P is *fully R^l -confluent* if it is R -confluent and for every derivative Q of P and $\rho \in R^\pm$, if $Q \uparrow \rho$ and $Q \xrightarrow{\rho} Q'$ then $Q' \uparrow$.

Thus if a derivative of a fully R^l -confluent agent diverges on an R -action it may not perform that action and reach a convergent state.

Lemma 4.1.13 Suppose P is fully R^l -confluent.

1. If $P \uparrow \rho$ and $P \xrightarrow{\rho} \simeq_1 P'$, where $\rho \in R^\pm$ then $P' \uparrow$.
2. If $P \uparrow s$ and $P \xrightarrow{s} \simeq_1 P'$, where $s \in R^{\pm*}$ then $P' \uparrow$.

PROOF: The proofs of these properties follow easily from the definitions of \simeq_1 and full R^l -confluence. We consider the proof of (2). Suppose $P \uparrow s$ and $P \xrightarrow{s} \simeq_1 P'$ for P a fully R^l -confluent agent. The proof is by induction on the length of s . If s is the empty sequence then $P \simeq_1 P'$ and so $P' \uparrow$ as required. So suppose $s = s_0\rho$ and $P \xrightarrow{s_0} \simeq_1 Q \xrightarrow{\rho} \simeq_1 P'$. There are two possibilities:

- If $P \uparrow s_0$ then by the induction hypothesis $Q \uparrow$. So $Q \uparrow \rho$ and by Property (1), $P' \uparrow$.
- If $P \downarrow s_0$ and $P \uparrow s$ then there must exist P_1, P_2 such that $P \xrightarrow{s_0} P_1 \downarrow \xrightarrow{\rho} P_2 \uparrow$. By Lemma 4.1.10(4), $Q \Rightarrow Q' \simeq P_1$ and by \simeq_1 we deduce that $Q' \uparrow \rho$. Therefore $Q \uparrow \rho$ and by the previous property we conclude that $P' \uparrow$. \square

We may now establish the first significant result. It implies that the state of a restricted composition of fully R^l -confluent agents is not changed up to db-bisimilarity by intra-actions on names of sort R , assuming that all R -names are restricted.

Theorem 4.1.14 Assume:

1. P and I are fully R^l -confluent;
2. $P \xrightarrow{s} \simeq_1 P'$, $I \xrightarrow{\bar{s}} \simeq_1 I'$, where $s = \rho_1 \dots \rho_n$, $\bar{s} = \bar{\rho}_1 \dots \bar{\rho}_n$, $\rho_i = r_i \langle \tilde{v}_i \rangle$, $\bar{\rho}_i = (\nu \tilde{u}_i) \bar{r}_i \langle \tilde{v}_i \rangle$ with $r_i : R$;
3. $(\nu \tilde{z})(P \mid I)$ is R -closed.

Then $(\nu \tilde{z})(P \mid I) \simeq_1 (\nu \tilde{z} \tilde{u})(P' \mid I')$, where $\tilde{u} = \tilde{u}_1 \dots \tilde{u}_n$.

PROOF: Let $(Q, Q') \in \mathcal{B}$ if $Q = (\nu \tilde{z})(P \mid I)$, $Q' = (\nu \tilde{z} \tilde{u})(P' \mid I')$, where P and I are fully R^l -confluent, $P \xrightarrow{s} \simeq_1 P'$, $I \xrightarrow{\bar{s}} \simeq_1 I'$, where $s = \rho_1 \dots \rho_n$, $\bar{s} = \bar{\rho}_1 \dots \bar{\rho}_n$, $\rho_i = r_i \langle \tilde{v}_i \rangle$, $\bar{\rho}_i = (\nu \tilde{u}_i) \bar{r}_i \langle \tilde{v}_i \rangle$ with $r_i : R$, $\tilde{u} = \tilde{u}_1 \dots \tilde{u}_n$ and such that $(\nu \tilde{z})(P \mid I)$ is R -closed. We show that $\mathcal{B} \cup \simeq_1$ is a db-bisimulation. The proof makes use of the following three results whose aim is to establish the divergence-related property required by \simeq_1 , namely that $Q \downarrow \beta$ iff $Q' \downarrow \beta$, for all $\beta \in \text{Act}$.

Lemma 4.1.15 Let $(Q, Q') \in \mathcal{B}$ where $Q' \downarrow$. Suppose $Q \xrightarrow{\tau} Q_1 = (\nu \tilde{z}')(P_1 \mid I_1)$, where $P_1 \downarrow$ and $I_1 \downarrow$. Then there exists Q'_1 such that $Q' \Rightarrow Q'_1$ and $(Q_1, Q'_1) \in \mathcal{B}$.

PROOF: Suppose $Q = (\nu \tilde{z})(P \mid I)$, $Q' = (\nu \tilde{z} \tilde{u})(P' \mid I')$, $P \xrightarrow{s} \simeq_1 P'$, $I \xrightarrow{\bar{s}} \simeq_1 I'$, where $s \in R^{+*}$ and $s \text{ comp } \bar{s}$. Note that since $Q' \downarrow$, $P' \downarrow$ and $I' \downarrow$ and so by Lemma 4.1.13(2), $P \downarrow s$ and $I \downarrow \bar{s}$. The following possibilities exist:

- $Q \xrightarrow{\tau} Q_1 = (\nu \tilde{z})(P_1 \mid I)$ where $P \xrightarrow{\tau} P_1$. Since $P \downarrow s$, by Lemma 4.1.10 (2), $P' \Rightarrow P'_1$ with $P_1 \xrightarrow{s} \simeq_1 P'_1$. So $Q' \Rightarrow Q'_1 = (\nu \tilde{z} \tilde{u})(P'_1 \mid I')$ as required.
- $Q \xrightarrow{\tau} Q_1 = (\nu \tilde{z})(P \mid I_1)$ where $I \xrightarrow{\tau} I_1$. This is similar to the previous case.
- $Q \xrightarrow{\tau} Q_1 = (\nu \tilde{z} \tilde{w})(P_1 \mid I_1)$ where $P \xrightarrow{\sigma} P_1$, $I \xrightarrow{\bar{\sigma}} I_1$, $\sigma \text{ comp } \bar{\sigma}$. Two cases exist:

1. First suppose $\sigma = r\langle \tilde{y} \rangle$, $\bar{\sigma} = (\nu \tilde{w})\bar{r}\langle \tilde{y} \rangle$, and $r \in \text{subj}(s)$. Suppose $s = s_0 \rho s_1$ and $\bar{s} = \bar{s}_0 \bar{\rho} \bar{s}_1$, where $r \notin \text{subj}(s_0)$ and $P \xrightarrow{s_0} \simeq_1 P_2 \xrightarrow{\rho} \simeq_1 P_3 \xrightarrow{s_1} \simeq_1 P'$, $I \xrightarrow{\bar{s}_0} \simeq_1 I_2 \xrightarrow{\bar{\rho}} \simeq_1 I_3 \xrightarrow{\bar{s}_1} \simeq_1 I'$. Then by Lemma 4.1.11(3) applied to I , $I_2 \Rightarrow \xrightarrow{\bar{\sigma}}$ and by R^1 -confluence $\bar{\sigma} = \bar{\rho}$. Hence, $I_1 \xrightarrow{\bar{s}/\bar{\sigma}} \simeq_1 I'$, by Lemma 4.1.11(4). Also $\sigma = \rho_i$ and by the same Lemma, $P_1 \xrightarrow{s/\sigma} \simeq_1 P'$. Thus $(Q_1, Q') \in \mathcal{B}$.
2. So suppose $\text{subj}(\sigma) \notin \text{subj}(s \bar{s})$. If $P \uparrow s\sigma$ then by Lemma 4.1.11(1), $P' \uparrow \sigma$ and $P' \Rightarrow \xrightarrow{\sigma} \uparrow$. On the other hand, if $P \downarrow s\sigma$ then by R^1 -confluence and Lemma 4.1.11(4), $P' \Rightarrow \xrightarrow{\sigma} P'_1$ with $P_1 \xrightarrow{s} \simeq_1 P'_1$. Similarly, if $I \uparrow \bar{s}\bar{\sigma}$ then by Lemma 4.1.11(1), $I' \uparrow \bar{\sigma}$, when since $I' \downarrow$, $I' \Rightarrow \xrightarrow{\bar{\sigma}} \uparrow$. Otherwise, if $I \downarrow \bar{s}\bar{\sigma}$ then by R^1 -confluence and Lemma 4.1.11(4), $I' \Rightarrow \xrightarrow{\bar{\sigma}} I'_1$ with $I_1 \xrightarrow{\bar{s}} \simeq_1 I'_1$. This implies that if either $P \uparrow s\sigma$ or $I \uparrow \bar{s}\bar{\sigma}$ then $Q' \Rightarrow \xrightarrow{\tau} \uparrow$ which gives a contradiction. So it must be that $P \downarrow s\sigma$ and $I \downarrow \bar{s}\bar{\sigma}$ and hence $Q' \Rightarrow Q'_1 = (\nu \tilde{z} \tilde{w} \tilde{u})(P'_1 \mid I'_1)$, where $(Q_1, Q'_1) \in \mathcal{B}$, as required. \square

This result is useful in proving the following.

Lemma 4.1.16 Suppose $(Q, Q') \in \mathcal{B}$. Then $Q \uparrow$ iff $Q' \uparrow$.

PROOF: Let $Q = (\nu \tilde{z})(P \mid I)$, $Q' = (\nu \tilde{z} \tilde{u})(P' \mid I')$, where $P \xrightarrow{s} \simeq_1 P'$, $I \xrightarrow{\bar{s}} \simeq_1 I'$. It is clear that $Q \Rightarrow \simeq_1 Q'$. Thus if $Q' \uparrow$ then $Q \uparrow$. So suppose $Q \uparrow$. If $P \uparrow s$ then since P is fully R^1 -confluent, $P' \uparrow$. Similarly if $I \uparrow \bar{s}$ then since I is fully R^1 -confluent, $I' \uparrow$. In either case $Q' \uparrow$. So assume $P \downarrow s$ and $I \downarrow \bar{s}$. Since $Q \uparrow$, for $Q_j = (\nu \tilde{z} \tilde{v}_1 \dots \tilde{v}_j)(P_j \mid I_j)$, either

1. $Q = Q_0 \xrightarrow{\tau} Q_1 \xrightarrow{\tau} \dots \xrightarrow{\tau} Q_k$ with $P_j \downarrow$, $I_j \downarrow$ for $j < k$ and $P_k \uparrow$ or $I_k \uparrow$, or

2. $Q = Q_0 \xrightarrow{\tau} Q_1 \xrightarrow{\tau} \dots$ with $P_j \downarrow, I_j \downarrow$ for $j < \omega$.

First consider the first possibility and suppose $Q' \downarrow$. We will derive a contradiction. Since $P \downarrow s$ and $I \downarrow \bar{s}$, by repeated application of the Lemma 4.1.15, $Q'_0 \Rightarrow Q'_1 \Rightarrow \dots \Rightarrow Q'_{k-1}$, where for $j < k$ $Q'_j = (\nu \tilde{z} \tilde{u} \tilde{v}_1 \dots \tilde{v}_j)(P'_j \mid I'_j)$, $(Q_j, Q'_j) \in \mathcal{B}$ and there exists $s_j \in R^{+*}$, $\bar{s}_j \in R^{-*}$, $s_j \text{ comp } \bar{s}_j$ such that $P_j \xrightarrow{s_j} \simeq_1 P'_j$ and $I_j \xrightarrow{\bar{s}_j} \simeq_1 I'_j$. Now consider the transition $Q_{k-1} \xrightarrow{\tau} Q_k$. Note that $P_{k-1} \downarrow$ and $I_{k-1} \downarrow$, whereas $P_k \uparrow$ or $I_k \uparrow$. This implies that the transition is a communication between the two components, that is, for some $\alpha, \bar{\alpha} \neq \tau$, $\alpha \text{ comp } \bar{\alpha}$, $P_{k-1} \xrightarrow{\alpha} P_k$ and $I_{k-1} \xrightarrow{\bar{\alpha}} I_k$. Since $P_{k-1} \downarrow s_{k-1}$ and $I_{k-1} \downarrow \bar{s}_{k-1}$ then by Lemma 4.1.11(2), $\alpha \notin s_{k-1}$. Suppose $P_k \uparrow$. Then $P_{k-1} \uparrow \alpha$, and by Lemma 4.1.11(2), $P_{k-1} \uparrow s_{k-1} \alpha$. Therefore by Lemma 4.1.11(1), $P'_{k-1} \uparrow \alpha$ and so $P'_{k-1} \xrightarrow{\alpha} P'_k \uparrow$. There are two cases for I_{k-1} : either $I_{k-1} \uparrow \bar{s}_{k-1} \bar{\alpha}$ or $I_{k-1} \downarrow \bar{s}_{k-1} \bar{\alpha}$. In the former case, since $I_{k-1} \downarrow \bar{s}_{k-1}$, $I'_{k-1} \uparrow \bar{\alpha}$, otherwise Lemma 4.1.11(1) would give a contradiction. So $I'_{k-1} \xrightarrow{\bar{\alpha}} I'_k$. Similarly, in the latter case, R^1 -confluence gives $I'_{k-1} \xrightarrow{\bar{\alpha}} I'_k$. This implies that $Q'_{k-1} \Rightarrow \xrightarrow{\tau} (\nu \tilde{z} \tilde{u} \tilde{v})(P'_k \mid I'_k) \uparrow$. This contradicts the assumption that $Q' \downarrow$. If $I_k \uparrow$ we can similarly derive a contradiction. Thus $Q' \uparrow$ as required.

On the other hand if the second possibility holds then since $P_j \downarrow, I_j \downarrow$ for all j , there are infinitely many i such that for some $\alpha_i, \bar{\alpha}_i$, $\alpha_i \text{ comp } \bar{\alpha}_i$, $P_i \xrightarrow{\alpha_i} P_{i+1}$, $I_i \xrightarrow{\bar{\alpha}_i} I_{i+1}$. By repeated application of the argument above it follows that there is a diverging computation from Q' . Hence $Q' \uparrow$. \square

We have the following corollary.

Corollary 4.1.17 Suppose $(Q, Q') \in \mathcal{B}$. Then for all $\beta \in \text{Act}$, $Q \uparrow \beta$ iff $Q' \uparrow \beta$.

PROOF: Let Q, Q' be as above. It is straightforward to see that if $Q' \uparrow \beta$ then also $Q \uparrow \beta$. So suppose that $Q \uparrow \beta$. If $Q \uparrow$ then the result follows from the previous lemma. So suppose $Q \downarrow \Rightarrow R \xrightarrow{\beta} S \uparrow$, where $R = (\nu \tilde{z} \tilde{w})(P_1 \mid I_1)$. If $Q' \uparrow$ then we are done, otherwise by Lemma 4.1.15, $Q' \Rightarrow R' \downarrow$, where $R' = (\nu \tilde{z} \tilde{w} \tilde{u})(P'_1 \mid I'_1)$, $P_1 \xrightarrow{s'} \simeq_1 P'_1$ and $I_1 \xrightarrow{\bar{s}'} \simeq_1 I'_1$ for some $s' = \rho_1 \dots \rho_n$, $\bar{s}' = \bar{\rho}_1 \dots \bar{\rho}_n$, $\rho_i = r_i \langle \tilde{v}_i \rangle$, $\bar{\rho}_i = (\nu \tilde{u}_i) \bar{r}_i \langle \tilde{v}_i \rangle$ with $r_i : R$. If $P_1 \uparrow s' \beta$ then by Lemma 4.1.11(1), $P'_1 \uparrow \beta$. Hence $R' \uparrow \beta$ and so $Q' \uparrow \beta$. Similarly, if $I_1 \uparrow \bar{s}' \beta$ then $Q' \uparrow \beta$. So suppose $P_1 \downarrow s' \beta$, $I_1 \downarrow \bar{s}' \beta$, $P_1 \xrightarrow{\beta} P_2$, $S = (\nu \tilde{z} \tilde{w})(P_2 \mid I_1)$, where β is $x \langle \tilde{y} \rangle$ or $(\nu \tilde{v}) \bar{x} \langle \tilde{y} \rangle$ and $\tilde{y} \cap \tilde{w} \tilde{z} = \emptyset$. Then by R^1 -confluence $P'_1 \Rightarrow \xrightarrow{\beta} P'_2$, $P_2 \xrightarrow{s} \simeq_1 P'_2$ and $R' \xrightarrow{\beta} S' = (\nu \tilde{z} \tilde{w} \tilde{v})(P'_2 \mid I'_1)$. Hence by Lemma 4.1.16, since $S \uparrow$ and $(S, S') \in \mathcal{B}$, we also have that $S' \uparrow$ and $Q' \uparrow \beta$ as required. The case $Q = (\nu \tilde{z} \tilde{w} - \tilde{z}')(P_2 \mid I_1)$, where $P_1 \xrightarrow{\beta'} P_2$, β' is $(\nu \tilde{v}) \bar{x} \langle \tilde{y} \rangle$, $(\tilde{y} - \tilde{v}) \cap \tilde{z} \tilde{w} = \tilde{z}'$ uses the same argument. Similarly, if $I_1 \xrightarrow{\beta} I_2$ we can show that $Q' \uparrow \beta$. \square

We may now proceed with the proof of the main theorem.

Proof of Theorem 4.1.14

Suppose $(S, S') \in \mathcal{B}$, where $S = (\nu\tilde{z})(P \mid I)$, $S' = (\nu\tilde{z}\tilde{u})(P' \mid I')$, $P \xrightarrow{\tau} \simeq_1 P'$, $I \xrightarrow{\bar{s}} \simeq_1 I'$ for some $s = \rho_1 \dots \rho_n$, $\bar{s} = \bar{\rho}_1 \dots \bar{\rho}_n$, $\rho_i = r_i \langle \tilde{v}_i \rangle$, $\bar{\rho}_i = (\nu\tilde{u}_i)\bar{r}_i \langle \tilde{v}_i \rangle$ with $r_i : R$, and such that $(\nu\tilde{z})(P \mid I)$ is R -closed. We show that $\mathcal{B} \cup \simeq_1$ is a db-bisimulation.

Suppose $S' \downarrow \alpha$, $S' \xrightarrow{\alpha} Q'$. Then by Corollary 4.1.17, $S \downarrow \alpha$ and as $S \xrightarrow{\tau} \simeq_1 S'$ it is straightforward to find Q, Q_0 such that $S \Rightarrow Q_0 \xrightarrow{\alpha} Q$ and $Q_0 \simeq_1 S'$, $(Q, Q') \in \mathcal{B}$. So suppose $S \downarrow \alpha$, $S \xrightarrow{\alpha} Q$. By Corollary 4.1.17, $S' \downarrow \alpha$ and the following cases exist. Note that by assumption, $\alpha \notin R^\pm$.

1. $\alpha = \tau$. It follows from Lemma 4.1.15 that $S' \Rightarrow Q'$, where $(Q, Q') \in \mathcal{B}$.
2. $Q = (\nu\tilde{z})(P_1 \mid I)$, where α is $x \langle \tilde{y} \rangle$ or $(\nu\tilde{w})\bar{x} \langle \tilde{y} \rangle$, where $\tilde{y} \cap \tilde{z} = \emptyset$ and $P \xrightarrow{\alpha} P_1$. Then by Lemma 4.1.11(3) for some P'' , $P' \Rightarrow \xrightarrow{\alpha} P''$ and $P_1 \xrightarrow{\rho} \simeq_1 P''$. So as $\tilde{u} \cap x\tilde{y} = \emptyset$, $S' \Rightarrow \xrightarrow{\alpha} Q' = (\nu\tilde{z}\tilde{u})(P'' \mid I')$ and moreover $(Q, Q') \in \mathcal{B}$.
3. $Q = (\nu\tilde{z} - \tilde{z}')(P_1 \mid I)$ where α is $(\nu\tilde{w}\tilde{z}')\bar{x} \langle \tilde{y} \rangle$ and $P \xrightarrow{\alpha'} P_1$, where α' is $(\nu\tilde{w})\bar{x} \langle \tilde{y} \rangle$, $(\tilde{y} - \tilde{w}) \cap \tilde{z} = \tilde{z}'$. Again by Lemma 4.1.11(3), $P' \Rightarrow \xrightarrow{\alpha} P''$ and $P_1 \xrightarrow{\rho} \simeq_1 P''$. Hence $S' \Rightarrow \xrightarrow{\alpha} Q' = (\nu\tilde{z}\tilde{u} - \tilde{z}')(P'' \mid I')$ and $(Q, Q') \in \mathcal{B}$.
4. If I acts alone the arguments are similar to those just given. □

The notion of sorts and sorting plays a significant rôle in the following results. The following definition isolates an important kind of association between two disjoint set of sorts, M and R . Extending earlier notation, we write M^+ (resp. M^-) for the set of input (resp. output) actions whose subject has a sort in M , and $M^\pm = M^- \cup M^+$.

Definition 4.1.18 Let M and R be disjoint sets of sorts. The sorting λ is an M, R -*sorting* if the following hold:

1. for each $M' \in M$ there is $R' \in R$ such that if $\tilde{i} \in \lambda(M')$ then, $\tilde{i} = \tilde{i}_0 R'$ and $R \cap \tilde{i}_0 = \emptyset$;
2. for each $R' \in R$, if $R' \in \tilde{i} \in \lambda(S)$ then $S \in M$.

Moreover, if $\alpha \in M^\pm$ we write $\text{obj}_R(\alpha) = r$ where $r \in \text{obj}(\alpha)$ and $r : R' \in R$. □

Thus, in the context of an M, R -sorting, if $\alpha \in M^\pm$ then the last component of the object of α is an R -name, which we denote by $\text{obj}_R(\alpha)$, and none of the remaining components are of sorts R . Further, M are the only sorts allowed by the sorting to

carry names of sorts R . We may extend the definition of ' R -confluence' to the case where R is a set of sorts and its associated lemmas continue to hold. This extension is employed in the following definition which aims to formalize the behaviour of certain classes of agents.

Definition 4.1.19 Suppose M and R are disjoint sets of sorts and let λ be an M, R -sorting. A derivation-closed set S of fully $R^!$ -confluent agents is (M^-, R^+) -tidy if there is a partition $\{S^{\tilde{r}} \mid \tilde{r} \text{ a finite set of } R\text{-names}\}$ of S , an (M^-, R^+) -tidy partition, such that:

1. if $P \in S^{\tilde{r}}$ and $P \xrightarrow{\alpha} P'$ where $\alpha \notin M^- \cup R^+$, then $P' \in S^{\tilde{r}}$;
2. if $P \in S^{\tilde{r}}$ and $P \xrightarrow{\mu} P'$, where $\mu \in M^-$ and $r = \text{obj}_R(\mu) \notin \tilde{r}$, then $P' \in S^{\tilde{r}, r}$;
3. if $P \in S^{\tilde{r}}$ and $P \xrightarrow{\rho} P'$, where $\rho \in R^+$, $r = \text{subj}(\rho) \in \tilde{r}$ and $P' \in S^{\tilde{r}-r}$.

Further, S is (M^-, R^+) -ready if it is (M^-, R^+) -tidy and

- 4a. if $P \in S^{\tilde{r}}$ and $\rho \in R^+$ with $\text{subj}(\rho) \in \tilde{r}$, then $P \xrightarrow{\rho}$.

The notions (M^+, R^-) -tidy and (M^+, R^-) -tidy partition are defined dually. We say that S is (M^+, R^-) -disciplined if it is (M^+, R^-) -tidy and

- 4b. if $P \in S^r$ (where r is a singleton) and $P \downarrow$, then $P \Rightarrow \xrightarrow{\rho}$ where $\text{subj}(\rho) = \bar{r}$.

□

This definition aims to capture a relationship between actions via names of the distinguished sets of sorts M and R . This relationship associates to each action of the form $(\nu \tilde{y}r)\overline{m}(\tilde{x}, r)$, where $m : M' \in M$ and $r : R' \in R$, the fresh name r . In this way, an (M^-, R^+) -tidy partition of a set S of fully $R^!$ -confluent agents divides its agents into classes $S^{\tilde{r}}$, where the index \tilde{r} records the names via which the agents of the class should receive answers. Thus by Condition 2, whenever an agent in the partition performs an output action with subject of sort M which involves a fresh R -name r , it enters the class where the name r is recorded in the class's index reflecting the capability of the resulting agent to perform an input via this name (Condition 3). Note that the only R^+ actions possible are those whose subject belongs to the class's index. Actions not in $M^- \cup R^+$ preserve the class of an agent (Condition 1).

Condition 4a stipulates that an agent must be able to engage immediately in any of its outstanding companion actions. Finally, Condition 4b requires that if an agent has an outstanding companion action then it may perform this action, possibly after some internal actions which however do not alter its \simeq_1 state. Note that

Condition 4a uses a universal quantification over actions of R^+ . On the other hand, in Condition 4b we have an existential quantification over R^- actions. However, since for all $P \in \mathcal{S}$, P is R^l -confluent, by Clause 2 in the definition of R^l -confluence, there exists a unique action possible.

An important point to note is Clause 2 in the definition of a (M^-, R^+) -tidy partition \mathcal{S} . This only imposes a requirement on those M^- actions μ , with $\text{obj}_R(\mu)$ a *new* name. This condition and its significance for the soundness of the theory will be investigated in Section 4.4.

The intention of the above definition is to provide conditions on agents which guarantee that when an agent is placed in certain contexts, its behaviour is indistinguishable from that of the agent obtained by pruning parts of the initial agent's state space. This is explained and stated formally in the theorem which follows the introduction of an important piece of notation. Recall that, given a labelled transition system \mathcal{T} and a subset W of its set of points, $\mathcal{T}[W]$ denotes the system obtained by removing all points not in W from \mathcal{T} and all arrows incident on such points (Definition 3.9.2).

Notation 4.1.20 If \mathcal{T} is an (M^+, R^-) -tidy set with an (M^+, R^-) -tidy partition $\{\mathcal{T}^r\}_{r \in R}$, we write $\mathcal{T}^{\leq 1} = \mathcal{T} \cup \{\mathcal{T}^r \mid |r| \leq 1\}$. Further, for every point $P \in \mathcal{T}^{\leq 1}$ we write P^b for the point of $\mathcal{T}[\mathcal{T}^{\leq 1}]$ corresponding to P . \square

Thus $\mathcal{T}^{\leq 1}$ contains the points of \mathcal{T} which have at most one outstanding companion action. We now have the main result.

Theorem 4.1.21 Suppose M and R are disjoint sets of sorts and λ is an M, R -sorting. Suppose \mathcal{P} is (M^-, R^+) -ready with (M^-, R^+) -tidy partition $\{\mathcal{P}^r\}_{r \in R}$, and \mathcal{I} is (M^+, R^-) -disciplined with (M^+, R^-) -tidy partition $\{\mathcal{I}^r\}_{r \in R}$. Suppose $P \in \mathcal{P}^0$, $I \in \mathcal{I}^0$ and $(\nu \tilde{p})(P \mid I)$ is M, R -closed. Then $(\nu \tilde{p})(P \mid I) \simeq_1 (\nu \tilde{p})(P \mid I^b)$.

PROOF: Let $(S_1, S_2) \in \mathcal{R}$ if $S_1 = (\nu \tilde{z})(P_1 \mid I_1)$, $S_2 = (\nu \tilde{z} \tilde{w})(P_2 \mid I_2^b)$, where $P_2 \in \mathcal{P}^{\leq 1}$, $I_2 \in \mathcal{I}^{\leq 1}$, $P_1 \xrightarrow{s} \simeq_1 P_2$, $I_1 \xrightarrow{\bar{s}} \simeq_1 I_2$, $s = \rho_1 \dots \rho_n \in R^{+*}$, $\bar{s} = \bar{\rho}_1 \dots \bar{\rho}_n \in R^{-*}$ with $\rho_i = r_i \langle \tilde{y}_i \rangle$, $\bar{\rho}_i = (\nu \tilde{w}_i) \bar{r}_i \langle \tilde{y}_i \rangle$, and $\tilde{w} = \tilde{w}_1 \dots \tilde{w}_n$, and S_1 is $R \cup M$ -closed. We show that $\mathcal{B} \cup \simeq_1$ is a db-bisimulation. The proof makes use of the following three results which establish the property required by \simeq_1 regarding the divergent behaviour of the agents.

Lemma 4.1.22 Let $(Q, Q') \in \mathcal{R}$ where $Q' \downarrow$. Suppose $Q \xrightarrow{\tau} Q_1 = (\nu \tilde{z}')(P_1 \mid I_1)$, where $P_1 \downarrow$ and $I_1 \downarrow$. Then there exists Q'_1 such that $Q' \Rightarrow Q'_1$ and $(Q_1, Q'_1) \in \mathcal{R}$.

PROOF: Suppose $Q = (\nu \tilde{z})(P \mid I)$, $Q' = (\nu \tilde{z} \tilde{v})(P' \mid I^b)$, where $P \xrightarrow{s} \simeq_1 P'$, $I \xrightarrow{\bar{s}} \simeq_1 I'$, $s = \rho_1 \dots \rho_n \in R^{+*}$, $\bar{s} = \bar{\rho}_1 \dots \bar{\rho}_n \in R^{-*}$, $\rho_i \text{ comp } \bar{\rho}_i$. We can carry

out the same case analysis as in Lemma 4.1.15 except for one possibility: Suppose $Q \xrightarrow{\tau} Q_1 = (\nu\tilde{z}\tilde{w})(P_1 \mid I_1)$, where $P \xrightarrow{\bar{\gamma}} P_1$ and $I \xrightarrow{\gamma} I_1$, $\gamma = m\langle\tilde{v}, r'\rangle$ and $\bar{\gamma} = (\nu\tilde{w})\bar{m}\langle\tilde{v}, r'\rangle$ where m is of a sort in M and $P' \in \mathcal{P}^r$, $I' \in \mathcal{I}^r$. In this case the transition $I^b \xrightarrow{\bar{\gamma}}$ cannot be found as $I' \in \mathcal{I}^r$. However, since $I' \in \mathcal{I}^r$, $I' \downarrow$ and \mathcal{I} is (M^+, R^-) -disciplined, $I' \Rightarrow \bar{\beta} I_2$, where $\text{subj}(\beta) = \bar{r}$, say $\bar{\beta} = (\nu\tilde{x})\bar{r}\langle\tilde{y}\rangle$. Also as \mathcal{P} is (M^-, R^+) -ready, $P' \in \mathcal{P}^r$ and $\text{subj}(\beta) = r$, where $\beta = r\langle\tilde{y}\rangle$, $P' \xrightarrow{\beta} P_2$. Note that since $Q' \downarrow$, $P_2 \downarrow$ and $I_2 \downarrow$. Suppose $P \uparrow s\beta\gamma$. Then by Lemma 4.1.11(1), $P_2 \uparrow \gamma$. Since $P_2 \downarrow$, $P_2 \xrightarrow{\gamma} \uparrow$. On the other hand, if $P \downarrow s\beta\gamma$ then by Lemma 4.1.11(3), $P_2 \Rightarrow \gamma P'_1$, $P_1 \xrightarrow{s\beta} \simeq_1 P'_1$. Similarly, if $I \uparrow \overline{s\beta\gamma}$ then by Lemma 4.1.11(1), $I_2 \uparrow \bar{\gamma}$. Since $I_2 \downarrow$, $I_2 \xrightarrow{\bar{\gamma}} \uparrow$. Otherwise, if $I \downarrow \overline{s\beta\gamma}$ then by Lemma 4.1.11(3), $I_2 \Rightarrow \bar{\gamma} I'_2$, $I_1 \xrightarrow{s\beta} \simeq_1 I'_1$. This implies that if either $P \uparrow s\beta\gamma$ or $I \uparrow \overline{s\beta\gamma}$ then $Q \Rightarrow \tau \uparrow$ which gives a contradiction. Hence $P \downarrow s\beta\gamma$, $I \downarrow \overline{s\beta\gamma}$, and $(\nu\tilde{z}\tilde{v})(P' \mid I^b) \Rightarrow \tau (\nu\tilde{z}\tilde{v}\tilde{x})(P_2 \mid I_2^b) \Rightarrow T(\nu\tilde{z}\tilde{v}\tilde{x}\tilde{w})(P'_1 \mid I_1^b)$ for some T . It is easy to see by construction that $(Q, T) \in \mathcal{R}$ and $(Q_1, (\nu\tilde{z}\tilde{v}\tilde{x}\tilde{w})(P'_1 \mid I_1^b)) \in \mathcal{R}$ as required. \square

Lemma 4.1.23 Suppose $(S_1, S_2) \in \mathcal{R}$. Then $S_1 \uparrow$ iff $S_2 \uparrow$.

The proof uses Lemma 4.1.22 and a similar argument to that in Lemma 4.1.16. \square

The lemma has the following corollary.

Corollary 4.1.24 Let $(S_1, S_2) \in \mathcal{R}$. Then for all $\beta \in \text{Act}$, $S_1 \uparrow \beta$ iff $S_2 \uparrow \beta$.

The proof is similar to that of Corollary 4.1.17. \square

We may now proceed with the proof of the main theorem.

Proof of Theorem 4.1.21

Suppose $(S_1, S_2) \in \mathcal{R}$, where $S_1 = (\nu\tilde{z})(P_1 \mid I_1)$, $S_2 = (\nu\tilde{z}\tilde{w})(P_2 \mid I_2^b)$, $P_1 \xrightarrow{s} \simeq_1 P_2$, $I_1 \xrightarrow{\bar{s}} \simeq_1 I_2$, $s = \rho_1 \dots \rho_n \in R^{+*}$ and $\bar{s} = \bar{\rho}_1 \dots \bar{\rho}_n \in R^{-*}$. Suppose $S_2 \downarrow \alpha$ and $S_2 \xrightarrow{\alpha} Q'_2$. Then by Corollary 4.1.24, $S_1 \downarrow \alpha$ and it is not difficult to see that $(\nu\tilde{z}\tilde{w})(P_2 \mid I_2) \xrightarrow{\alpha} Q_2$ for some Q_2 with $(Q_2, Q'_2) \in \mathcal{R}$. By Theorem 4.1.14, $S_1 \simeq_1 (\nu\tilde{z}\tilde{w})(P_2 \mid I_2)$, so $S_1 \Rightarrow \alpha Q_1$ with $Q_1 \simeq_1 Q_2$. Thus $(Q_1, Q'_2) \in \simeq_1 \mathcal{R}$. Now suppose $S_1 \downarrow \alpha$ and $S_1 \xrightarrow{\alpha} Q_1$. By Corollary 4.1.24, $S_1 \downarrow \alpha$. There are several cases.

1. Suppose $\alpha = \tau$. Then by Lemma 4.1.22 it follows that $S_2 \Rightarrow Q_2$ with $(Q_1, Q_2) \in \mathcal{R}$.
2. Suppose $Q_1 \equiv (\nu\tilde{z})(P'_1 \mid I_1)$ where $P_1 \xrightarrow{\alpha} P'_1$ and α is $x\langle\tilde{y}\rangle$ or $(\nu\tilde{u})\bar{x}\langle\tilde{y}\rangle$, where $\tilde{u} \cap \tilde{z} = \emptyset$. Then by Lemma 4.1.11(3) there are P'_2, P''_2 such that $P_2 \Rightarrow P'_2 \xrightarrow{\alpha} P''_2$ and $P'_1 \xrightarrow{s} \simeq_1 P''_2$ with $P'_2 \simeq_1 P_2$. So $S_2 \Rightarrow (\nu\tilde{z}\tilde{w})(P'_2 \mid I_2^b) \xrightarrow{\alpha}$

$(\nu\tilde{z}\tilde{w})(P_2'' \mid I_2^b)$, and it is easy to see that $(S_1, (\nu\tilde{z}\tilde{w})(P_2' \mid I_2^b)) \in \mathcal{R}$ and $(Q_1, (\nu\tilde{z}\tilde{w})(P_2'' \mid I_2^b)) \in \mathcal{R}$.

3. Suppose $Q_1 \equiv (\nu\tilde{z})(P_1 \mid I_1')$ where $I_1 \xrightarrow{\alpha} I_1'$. The proof is similar to the previous case. \square

According to Theorem 4.1.21, an (M^+, R^-) -disciplined agent I is indistinguishable from its pruned version I^b within a context $\mathcal{C}[\cdot] = (\nu\tilde{z})(P \mid \cdot)$ where P is (M^-, R^+) -ready. That is, $\mathcal{C}[I] \simeq_1 \mathcal{C}[I^b]$. The import of this result is that when reasoning about agent $\mathcal{C}[I]$, we may restrict our attention to the fragment of its behaviour.

Finally, we consider a special case of Theorem 4.1.21 involving a subclass of (M^-, R^+) -ready partitions. In particular, we are interested in partitions whose agents may emit only *new* R -names.

Definition 4.1.25 Let R be a set of sorts. An agent P is *R -polite* if, for all derivatives P' of P , whenever $P' \xrightarrow{(\nu\tilde{y})\tilde{a}(\tilde{x})}$ and $x_i : R$, then $x \in \tilde{y}$.

We observe that in order to prove that an agent I is indistinguishable from its pruned version within an R -polite, (M^-, R^+) -ready context, it is sufficient to ensure that only a part of the transition system generated by I, I^R , defines a (M^+, R^-) -tidy partition.

Theorem 4.1.26 Suppose M and R are disjoint sets of sorts and λ is an M, R -sorting. Suppose \mathcal{P} is (M^-, R^+) -ready with (M^-, R^+) -tidy partition $\{\mathcal{P}^r\}_{\tilde{r}}$, and \mathcal{I}^R is (M^+, R^-) -disciplined with (M^+, R^-) -tidy partition $\{\mathcal{I}^r\}_{\tilde{r}}$. Suppose $P \in \mathcal{P}^\emptyset$, P is R -polite, $I \in \mathcal{I}^\emptyset$ and $(\nu\tilde{p})(P \mid I)$ is M, R -closed. Then $(\nu\tilde{p})(P \mid I) \simeq_1 (\nu\tilde{p})(P \mid I^b)$.

PROOF: It is easy to see that since P is R -polite,

$$(\nu\tilde{p})(P \mid I) \simeq_1 (\nu\tilde{p})(P \mid I^R).$$

Furthermore, by Theorem 4.1.21,

$$(\nu\tilde{p})(P \mid I^R) \simeq_1 (\nu\tilde{p})(P \mid (I^R)^b).$$

Since I^b (resp. $(I^R)^b$) represents the transition system of I (resp. (I^R)) which has at most one outstanding companion action, it is easy to see that the transition systems corresponding to I^b and $(I^R)^b$ coincide: $I^b \simeq_1 (I^R)^b$. Thus, $(\nu\tilde{p})(P \mid (I^R)^b) \simeq_1 (\nu\tilde{p})(P \mid I^b)$ and the result follows. \square

4.2 db^R -bisimilarity

In this section we consider a variant of db-bisimilarity which restricts attention to divergence with respect to a certain set of actions. The main result states that two agents related by this weaker relation are indistinguishable in certain contexts. We begin by introducing a new divergence relation.

Notation 4.2.1 Let $R \subseteq \text{Act}$. We write $P \downarrow R$ if $P \downarrow \rho$ for all $\rho \in R$. We write $P \uparrow R$ if not $P \downarrow R$.

Hence an agent converges on a set of actions, R , if it is convergent, and it remains convergent after any visible action via a name in R .

Definition 4.2.2 Let $R \subseteq \text{Act}$. The relation db^R -bisimilarity, \simeq_1^R , is the largest such that, if $P \simeq_1^R Q$ then

1. if $P \downarrow R$ then $Q \downarrow R$ and for all α with $\text{bn}(\alpha) \cap \text{fn}(P, Q) = \emptyset$, if $P \xrightarrow{\alpha} P'$ then
 - (a) $Q \Rightarrow Q'' \xrightarrow{\alpha} Q'$ with $P \simeq_1^R Q''$ and $P' \simeq_1^R Q'$, or
 - (b) $\alpha = \tau$ and $P' \simeq_1^R Q$, and
2. vice versa.

Thus if two agents are db^R -bisimilar then each may mimic the other's behaviour until one of them becomes divergent or may become divergent by performing an R -action. Moreover, if one of the agents is divergent on R then so is the other one. It is easy to see that \simeq_1 and \simeq_1^R are unrelated: for $P \stackrel{\text{def}}{=} r.\Omega + a.0$, $Q \stackrel{\text{def}}{=} r.\Omega + a.0 + b.0$, $r \in R$, we have that $P \simeq_1^R Q$ but $P \not\simeq_1 Q$ as $Q \downarrow b$, $Q \xrightarrow{b}$ but $P \not\xrightarrow{b}$, and $a.\Omega \simeq_1 a.\Omega + a.0$ whereas $a.\Omega \not\simeq_1^R a.\Omega + a.0$.

Given a set of sorts R , the following theorem states that two db^{R^-} -bisimilar agents belonging to the same block of an (M^+, R^-) -disciplined partition are indistinguishable up to db-bisimilarity within a (M^-, R^+) -ready context.

Theorem 4.2.3 Suppose λ is an M, R -sorting. Suppose \mathcal{P} is an (M^-, R^+) -ready system with (M^-, R^+) -tidy partition $\{\mathcal{P}^{\tilde{r}}\}_{\tilde{r}}$, and \mathcal{I} an (M^+, R^-) -disciplined system with (M^+, R^-) -tidy partition $\{\mathcal{I}^{\tilde{r}}\}_{\tilde{r}}$. Suppose $P \in \mathcal{P}^{\tilde{r}}$, $I_1, I_2 \in \mathcal{I}^{\tilde{r}}$, $I_1 \simeq_1^{R^-} I_2$ and $(\nu \tilde{p})(P \mid I_1)$, $(\nu \tilde{p})(P \mid I_2)$ are M, R -closed. Then $(\nu \tilde{p})(P \mid I_1) \simeq_1 (\nu \tilde{p})(P \mid I_2)$.

PROOF: Let $(S_1, S_2) \in \mathcal{B}$ if $S_1 = (\nu \tilde{p})(P \mid I_1)$ and $S_2 = (\nu \tilde{p})(P \mid I_2)$, where $P \in \mathcal{P}^{\tilde{r}}$, $I_1, I_2 \in \mathcal{I}^{\tilde{r}}$, $I_1 \simeq_1^{R^-} I_2$. We show that $\mathcal{B} \cup \simeq_1$ is a db-bisimulation. The proof makes use of the following three results.

Lemma 4.2.4 Let $(Q, Q') \in \mathcal{B}$ where $Q = (\nu \tilde{p})(P \mid I)$, $I \downarrow R^-$. Suppose $Q \xrightarrow{\tau} Q_1$. Then there exists Q'_1 such that $Q' \Rightarrow Q'_1$ and $(Q_1, Q'_1) \in \mathcal{B}$.

PROOF: Suppose $Q' = (\nu \tilde{p})(P \mid I')$ where $I \dot{\simeq}_1^{R^-} I'$, $P \in \mathcal{P}^{\tilde{r}}$ and $I, I' \in \mathcal{I}^{\tilde{r}}$. The following possibilities exist:

- $Q \xrightarrow{\tau} Q_1 = (\nu \tilde{p})(P_1 \mid I)$ where $P \xrightarrow{\tau} P_1$. Clearly, $Q' \xrightarrow{\tau} Q'_1 = (\nu \tilde{p})(P_1 \mid I')$ and $(Q_1, Q'_1) \in \mathcal{B}$.
- $Q \xrightarrow{\tau} Q_1 = (\nu \tilde{p})(P \mid I_1)$ where $I \xrightarrow{\tau} I_1$. Since $I \downarrow R^-$ and $I \dot{\simeq}_1^{R^-} I'$, either $I_1 \dot{\simeq}_1^{R^-} I'$ and $(Q_1, Q') \in \mathcal{B}$, or $I' \Rightarrow \xrightarrow{\tau} I'_1$ where $I_1 \dot{\simeq}_1^{R^-} I'_1$ and $Q' \Rightarrow \xrightarrow{\tau} Q'_1 = (\nu \tilde{p})(P \mid I'_1)$ with $(Q_1, Q'_1) \in \mathcal{B}$ as required.
- $Q \xrightarrow{\tau} Q_1 = (\nu \tilde{p}\tilde{u})(P_1 \mid I_1)$, $P \xrightarrow{\alpha} P_1$, $I \xrightarrow{\bar{\alpha}} I_1$ where $\alpha \text{ comp } \bar{\alpha}$ and $\tilde{u} = \text{bn}(\alpha) \cup \text{bn}(\bar{\alpha})$. Since $I \downarrow R^-$ and $I \dot{\simeq}_1^{R^-} I'$, $I' \Rightarrow \xrightarrow{\bar{\alpha}} I'_1$ where $I_1 \dot{\simeq}_1^{R^-} I'_1$. Hence $Q' \Rightarrow \xrightarrow{\tau} Q'_1 = (\nu \tilde{p}\tilde{u})(P' \mid I'_1)$ where $(Q'_1, Q'_1) \in \mathcal{B}$ as required. Note in particular that, if $\alpha \in M^-$ then $P_1 \in \mathcal{P}^{\tilde{r}, r}$, where $r = \text{obj}_R(\alpha)$, $I_1, I'_1 \in \mathcal{I}^{\tilde{r}, r}$, whereas, if $\alpha \in R^+$ then $P_1 \in \mathcal{P}^{\tilde{r}-r}$, $I_1, I'_1 \in \mathcal{I}^{\tilde{r}-r}$, where $r = \text{subj}(\alpha)$. \square

Lemma 4.2.5 Suppose $(Q, Q') \in \mathcal{B}$. Then $Q \uparrow$ iff $Q' \uparrow$.

PROOF: Suppose $Q = (\nu \tilde{p})(P \mid I)$, $Q' = (\nu \tilde{p})(P \mid I')$, where P, I and I' are as above. First we note that, if $I \uparrow R^-$ (resp. $I' \uparrow R^-$) then $Q \uparrow$ (resp. $Q' \uparrow$): if $I \uparrow$ then clearly, $Q \uparrow$. Otherwise, $I \Rightarrow \xrightarrow{\bar{\rho}} I_1 \uparrow$, for some $\bar{\rho} \in R^-$, $\text{subj}(\rho) \in \tilde{r}$, where $I \in \mathcal{I}^{\tilde{r}}$. Since $P \in \mathcal{P}^{\tilde{r}}$ and P is (M^-, R^+) -ready, $P \xrightarrow{\rho} P_1$, $\rho \text{ comp } \bar{\rho}$ and $Q \Rightarrow \xrightarrow{\tau} Q_1$.

Suppose $Q \uparrow$. If $P \uparrow$ then also $Q' \uparrow$. On the other hand, if $I \uparrow R^-$ then by definition of $\dot{\simeq}_1^{R^-}$, $I' \uparrow R^-$. By the observation above, $Q' \uparrow$. So assume $P \downarrow$ and $I \downarrow R^-$. Since $Q \uparrow$, for $Q_j = (\nu \tilde{p}_j)(P_j \mid I_j)$, either

1. $Q = Q_0 \xrightarrow{\tau} Q_1 \xrightarrow{\tau} \dots \xrightarrow{\tau} Q_k$ with $P_j \downarrow$, $I_j \downarrow R^-$ for $j < k$ and $P_k \uparrow$ or $I_k \uparrow R^-$, or
2. $Q = Q_0 \xrightarrow{\tau} Q_1 \xrightarrow{\tau} \dots$ with $P_j \downarrow$, $I_j \downarrow R^-$ for $j < \omega$.

First consider (1). Then by repeated application of the previous lemma $Q'_0 \Rightarrow Q'_1 \Rightarrow \dots \Rightarrow Q'_{k-1}$ where for $j < k$, $(Q_j, Q'_j) \in \mathcal{B}$ and $Q'_j = (\nu \tilde{p}_j)(P_j \mid I'_j)$ where $I_j \dot{\simeq}_1^{R^-} I'_j$. Now consider the transition $Q_{k-1} \xrightarrow{\tau} Q_k$. Note that for $j < k$, $P_j \downarrow$ and $I_j \downarrow R^-$, whereas either $P_k \uparrow$ or $I_k \uparrow R^-$. This implies that the transition is a communication between the two components, that is, for some α , $P_{k-1} \xrightarrow{\alpha} P_k$ and $I_{k-1} \xrightarrow{\bar{\alpha}} I_k$, where $\alpha \text{ comp } \bar{\alpha}$. Suppose $I_k \uparrow R^-$. Since $I'_{k-1} \downarrow R^-$ then by definition of $\dot{\simeq}_1^{R^-}$, $I'_{k-1} \Rightarrow \xrightarrow{\bar{\alpha}} I'_k$, $I_k \dot{\simeq}_1^{R^-} I'_k$. Since $I_k \uparrow R^-$, $I'_k \uparrow R^-$ and by the

observation above we conclude that $Q'_{k-1} \Rightarrow \xrightarrow{\tau} (\nu \widetilde{p_k})(P_k \mid I'_k) \uparrow$. Similarly, if $P_k \uparrow$ then $Q'_{k-1} \Rightarrow \xrightarrow{\tau} (\nu \widetilde{p_k})(P_k \mid I'_k) \uparrow$ as required. The converse is similar.

If (2) holds then since $P_j \downarrow, I_j \downarrow R^-$ for all j , there are infinitely many i such that for some $\alpha_i, P_i \xrightarrow{\alpha_i} P_{i+1}, I_i \xrightarrow{\bar{\alpha}_i} I_{i+1}$, where $\alpha_i \text{ comp } \bar{\alpha}_i$. By repeated application of the argument above it follows that there is a diverging computation from Q' . Hence $Q' \uparrow$. \square

Corollary 4.2.6 Suppose $(Q, Q') \in \mathcal{B}$. Then for all $\beta \in \text{Act}$, $Q \uparrow \beta$ iff $Q' \uparrow \beta$.

PROOF: Let Q, Q' be as above. If $Q \uparrow$ then the result follows from the previous lemma. So suppose $Q \downarrow \Rightarrow T \xrightarrow{\beta} S \uparrow$ where $T = (\nu \widetilde{p})(P \mid I_1)$. If $Q' \uparrow$ then we are done; otherwise, since $I \downarrow R^-$ as $Q \downarrow$, by Lemma 4.2.4, $Q' \Rightarrow T' \downarrow$, where $T' = (\nu \widetilde{p})(P \mid I'_1)$, $(T, T') \in \mathcal{B}$. If $P \uparrow \beta$ then clearly $T' \uparrow \beta$ and the result follows. Otherwise, if $I_1 \uparrow \beta$ then $I_1 \xrightarrow{\beta} I_2 \uparrow$. By the observation in the proof of Lemma 4.2.5, since $T \downarrow, I_1 \downarrow R^-$. So by $\simeq_1^{R^-}$, $I'_1 \Rightarrow \xrightarrow{\beta} I'_2 \simeq_1^{R^-} I_2$. Since $I_2 \uparrow, I'_2 \uparrow R^-$. Using the observation above we conclude that $(\nu \widetilde{p})(P \mid I'_2) \uparrow$ and so $Q' \Rightarrow T' \Rightarrow \xrightarrow{\beta} \uparrow$ as required. The converse is similar. \square

The proof of the theorem now follows.

Proof of Theorem 4.2.3

Let $(S, S') \in \mathcal{B}$, where $S = (\nu \widetilde{p})(P \mid I)$ and $S' = (\nu \widetilde{p})(P \mid I')$. Suppose $S \downarrow \alpha$, $S \xrightarrow{\alpha} Q$. Then by Corollary 4.2.6, $S' \downarrow \alpha$. Note that since $S \downarrow \alpha, S \downarrow$ and thus $P \downarrow$ and $I \downarrow R^-$. We have to consider the following cases:

1. $\alpha = \tau$. It follows from Lemma 4.2.4 that $S' \Rightarrow Q'$ where $(Q, Q') \in \mathcal{B}$.
2. $P \xrightarrow{\alpha} P_1, Q = (\nu \widetilde{p})(P_1 \mid I)$. Clearly, $S' \xrightarrow{\alpha} Q' = (\nu \widetilde{p})(P_1 \mid I')$, and $(Q, Q') \in \mathcal{B}$.
3. $I \xrightarrow{\alpha} I_1, Q = (\nu \widetilde{p})(P \mid I_1)$. Since $I \downarrow R^-$ and $I \simeq_1^{R^-} I', I' \Rightarrow \xrightarrow{\alpha} I'_1 \simeq_1^{R^-} I_1$. Hence $Q' \Rightarrow \xrightarrow{\alpha} Q'_1 = (\nu \widetilde{p})(P \mid I'_1)$, where $(Q_1, Q'_1) \in \mathcal{B}$ as required.

The converse is similar. \square

4.3 Social confluence

The main result of Section 4.1, Theorem 4.1.21, states that in an (M^-, R^+) -ready context, an (M^+, R^-) -disciplined agent I is indistinguishable from its fragment I^b . An important requirement imposed on an (M^+, R^-) -disciplined agent is that on accepting a question via an M -name, it immediately assumes a state in which the

answer to that question is determined, and it is capable of returning that answer by performing an R^- -action after some silent actions which however do not change its state up to branching bisimilarity.

The form of interaction captured by (M^-, R^+) -disciplined and (M^+, R^-) -ready agents is very common in computer systems, for example in shared memory and database systems. Furthermore, the main result appears to be closely related to the notions of correctness (sequential consistency, serializability) associated with such systems: it amounts to saying that within a ready context, a disciplined system is indistinguishable from its serial version, that is, a system supporting the same questions/operations but capable of processing at most one operation at a time. However, it is often the case that the ‘disciplined’ component of such a system does not satisfy the requirement above: the processing of a question by the component may result in a change of its state. For example the commitment to perform a write request of a register in a multiple-read single-write memory alters the state, as it resolves a choice by performing the request in question and discarding all behaviours corresponding to different operations actually taking place. The question then arises under which conditions such interactions may be serialized without altering a system’s observable behaviour. In this section we consider a generalization of the theory where agents of interest may perform up to one state-changing internal action before responding to M^+ actions via the companion R^- actions. It turns out that a number of extra conditions need to be imposed in order to carry the results to this setting.

In the presentation we distinguish two sets of sorts Q (for ‘question’) and A (for ‘answer’), as opposed to M and R . We do not take divergence into account, and we work with the definition of R -confluence of [LW95a], presented in Section 4.1, and branching bisimilarity. This is appropriate for the application we consider in the next chapter: although divergence is present in the system in question in the form of livelock, the appropriate notion of correctness is branching bisimilarity. However, it is expected that the results can be extended to the divergence-sensitive setting using arguments similar to those of the previous sections. We begin with the following central definitions.

Definition 4.3.1 Let Q and A be disjoint sets of sorts. A process P is Q, A -socially confluent if it is Q -confluent, A -confluent and for every derivative R of P the following hold:

1. if $R \xrightarrow{\alpha} \simeq R_1 \xrightarrow{\beta} \simeq R_2$ where either $\alpha \notin Q^\pm \cup \{\tau\}$ or $\beta \notin A^\pm \cup \{\tau\}$, then $R \xrightarrow{\beta} \simeq R'_1 \xrightarrow{\alpha} \simeq R'_2$ where $R_2 \simeq R'_2$.

Thus a Q, A -social confluent agent must be partial confluent with respect to both sorts Q and A , and satisfy a further condition, which captures that certain actions commute. For example, the order in which two questions are asked or two answers given is not significant: the computations corresponding to each of the orderings lead to branching-bisimilar states. However, an answer may not be given before the corresponding question has been posed, and the order of two decisive internal actions may not be changed without change in behaviour.

Definition 4.3.2 Suppose Q and A are disjoint sets of sorts and let λ be a Q, A -sorting. A derivation-closed set \mathcal{R} of Q, A -socially confluent agents is a (Q^+, A^-) -base if there is a partition, a (Q^+, A^-) -base partition, $\{\mathcal{R}^{\tilde{a}} \mid \tilde{a} \text{ a finite set of } A\text{-names}\}$ of \mathcal{R} which is (Q^+, A^-) -tidy. The notions of a (Q^-, A^+) -base and a (Q^-, A^+) -base partition are defined dually.

Thus a (Q^+, A^-) -base partition is a (Q^+, A^-) -tidy partition of a set of Q, A -socially confluent agents. It is convenient to introduce the following definition.

Definition 4.3.3 An agent P is S -inert, for $S \subseteq \text{Act}$, if $\text{not}(P \Rightarrow \xrightarrow{\alpha})$ for all $\alpha \in S$.

Definition 4.3.4 We say that a derivation-closed set \mathcal{R} is a (Q^+, A^-) -server if there is a partition, a (Q^+, A^-) -server partition, $\{\mathcal{R}^{\tilde{a}} \mid \tilde{a} \text{ a finite set of } A\text{-names}\}$ of \mathcal{R} which is a (Q^+, A^-) -base partition and such that

2. if $R \in \mathcal{R}^a$ (where a is a singleton) then either $R \Rightarrow \xrightarrow{\alpha}$ or $R \Rightarrow \xrightarrow{\tau} R' \Rightarrow \xrightarrow{\alpha}$ for some $\alpha = \bar{a}(\tilde{v})$;
3. if $R \in \mathcal{R}^{\tilde{a}}$, R is A^- -inert and $R \xrightarrow{\tau} \simeq R'$, then there exists $\alpha \in A^-$ such that $R' \Rightarrow \xrightarrow{\alpha} R''$ and R'' is A^- -inert;
4. if $R \in \mathcal{R}^{\emptyset}$, $R \xrightarrow{u} \simeq \xrightarrow{\gamma} \simeq R_1$ where $\gamma \in \{\tau\} \cup A^-$ and $u \in Q^{+*}$, then there exists $\beta \in u$ such that $R \xrightarrow{\beta} \simeq \xrightarrow{\gamma} \simeq R_2$ and $R_2 \xrightarrow{u/\beta} \simeq R_1$;
5. if $R \in \mathcal{R}^a$, $R \xrightarrow{\tau} \simeq R_1$ and $R \xrightarrow{\tau} \simeq R_2$, then $R_1 \simeq R_2$.

Thus, the definition of a (Q^+, A^-) -server extends Clause 3b in the definition of a (M^+, R^-) -disciplined partition by requiring: in Condition 2, that an answer to an outstanding question is either already available or may become available after a single decisive τ -action; in Condition 3 that a decisive silent action leads to the determination of a result that was not previously available; in Condition 4, that there exists a unique question responsible for the occurrence of every decisive silent

action and every answer; and in Condition 5, that a decisive silent action results in a uniquely determined state. We have the following lemma.

Lemma 4.3.5 Suppose \mathcal{R} is a (Q^+, A^-) -server with (Q^+, A^-) -server partition $\{\mathcal{R}^a\}_a$. The following hold:

- i. If $R \in \mathcal{R}^\emptyset$ then R is $A^- \cup \{\tau\}$ -inert.
- ii. If $R \in \mathcal{R}^a$ is not A^- -inert then R is τ -inert.
- iii. If $R \in \mathcal{R}^\emptyset$ and $R \xrightarrow{u} \simeq R_1 \xrightarrow{\beta} \simeq \xrightarrow{\alpha} \simeq R_2$ where R_1 is A^- -inert, $\alpha \in A^-$ and $u\beta \in Q^{+*}$, then R_2 is A^- -inert.

PROOF: Property (i) is straightforward from the definition of a (Q^+, A^-) -tidy partition and Definition 4.3.4(3).

To prove (ii) suppose that $R \Rightarrow \xrightarrow{\alpha} R_1$ where $\alpha \in A^-$. If $R \Rightarrow \xrightarrow{\tau}$ then as R is A -confluent, $R_1 \Rightarrow \xrightarrow{\tau}$. But this contradicts (i) as $R_1 \in \mathcal{R}^\emptyset$.

Finally consider (iii). Suppose $R \in \mathcal{R}^\emptyset$ and $R \xrightarrow{u} \simeq R_1 \xrightarrow{\beta} \simeq \xrightarrow{\alpha} \simeq R_2$ where R_1 is A^- -inert, $\alpha \in A^-$ and $u\beta \in Q^{+*}$. For the sake of contradiction assume that R_2 is not A^- -inert and $R_2 \Rightarrow \xrightarrow{\alpha'} R'_2$, $\alpha' \in A^-$. By Definition 4.3.4(4),

$$R \xrightarrow{\gamma} \simeq \xrightarrow{\alpha} \simeq R' \text{ where } R' \xrightarrow{u\beta/\gamma} \simeq R_2$$

and $\gamma \in u\beta$. Suppose $\gamma \in u$ then by A -confluence and Lemma 4.1.5, $R_1 \Rightarrow \xrightarrow{\alpha}$ which contradicts the assumption that R_1 is A^- -inert. So it must be that $\beta = \gamma$. Since $R' \xrightarrow{u} \simeq R_2$ and $R_2 \Rightarrow \xrightarrow{\alpha'} R'_2$, by Definition 4.3.4(4)

$$R' \xrightarrow{\beta'} \simeq \xrightarrow{\alpha'} \simeq R'' \text{ and } R'' \xrightarrow{u/\beta'} \simeq R'_2$$

for some $\beta' \in u$. So we have that

$$R \xrightarrow{\beta} \simeq \xrightarrow{\alpha} \simeq R' \xrightarrow{\beta'} \simeq \xrightarrow{\alpha'} \simeq R''.$$

By Definition 4.3.1(1),

$$R \xrightarrow{\beta'} \simeq R_m \xrightarrow{\beta} \simeq \xrightarrow{\alpha} \simeq \xrightarrow{\alpha'} \simeq R''$$

and it is easy to see that R_m is A^- -inert (otherwise, since R_m is Q -confluence and A -confluent, $R_1 \Rightarrow \xrightarrow{\alpha}$ which contradicts the assumption of R_1 being A^- -inert). So by Definition 4.3.4(2),

$$R_m \xrightarrow{\tau} \simeq R'_m \xrightarrow{\alpha''} \simeq \text{ where } \alpha'' \in A^-.$$

Thus by Q -confluence and A -confluence,

$$R'_m \xrightarrow{\beta} \simeq \xrightarrow{\alpha} \simeq \xrightarrow{\alpha'} \simeq R''_m \text{ and } R'' \xrightarrow{\tau} \simeq R''_m.$$

Since $R'' \in \mathcal{R}^\emptyset$ by (i) this is a contradiction. Thus R_2 is A^- -inert as required. \square

Definition 4.3.6 A derivation-closed set \mathcal{E} is a (Q^-, A^+) -client if there is a partition, a (Q^-, A^+) -client partition, $\{\mathcal{R}^{\tilde{a}} \mid \tilde{a} \text{ a finite set of } A\text{-names}\}$ of \mathcal{R} which is a (Q^+, A^-) -base partition and such that

6. if $E \in \mathcal{E}^{\tilde{a}}$ and $\alpha \in A^+$ where $\text{subj}(\alpha) \in \tilde{a}$ then $E \xrightarrow{\alpha}$.

Thus a (Q^-, A^+) -client partition is obtained from a (Q^-, A^+) -base partition as a (P^-, Q^+) -ready partition is obtained from a (P^-, Q^+) -tidy partition. The following theorem implies that the state of a restricted composition of a client process and the pruned part of a server process is not altered up to branching bisimilarity by an intra-action via a name of a sort in A .

Theorem 4.3.7 Suppose λ is a Q, A -sorting. Further suppose \mathcal{R} is a (Q^+, A^-) -server with a (Q^+, A^-) -server partition $\{\mathcal{R}^{\tilde{a}}\}_{\tilde{a}}$ and \mathcal{E} is a (Q^-, A^+) -client with (Q^-, A^+) -client partition $\{\mathcal{E}^{\tilde{a}}\}_{\tilde{a}}$. Suppose $R_1 \in \mathcal{R}^{\tilde{a}}$, $E_1 \in \mathcal{E}^{\tilde{a}}$ and $R_1 \xrightarrow{\bar{\alpha}} R_2$, $E_1 \xrightarrow{\alpha} E_2$ where $\text{subj}(\alpha) = a$, $\alpha \text{ comp } \bar{\alpha}$. Further suppose $(\nu \tilde{z})(E_1 \mid R_1^b)$ is Q, A -closed. Then $(\nu \tilde{z})(E_1 \mid R_1^b) \simeq (\nu \tilde{z} \tilde{u})(E_2 \mid R_2^b)$, where $\tilde{u} = \text{bn}(\bar{\alpha})$.

PROOF: Note that since $R_1 \in \mathcal{R}^{\tilde{a}}$ and \mathcal{R} is a server system, by Definition 4.3.4(2) and (3), either $R_1 \xRightarrow{\bar{\alpha}} R_2$ or $R_1 \xRightarrow{\tau} R_2 \xRightarrow{\bar{\alpha}}$. Let $(S_1, S_2) \in \mathcal{B}_0$ if $S_1 = (\nu \tilde{z})(E_1 \mid R_1^b)$ and $S_2 = (\nu \tilde{z} \tilde{u})(E_2 \mid R_2^b)$ where $E_1 \in \mathcal{E}^{\tilde{a}}$, $R_1 \in \mathcal{R}^{\tilde{a}}$, $E_1 \xrightarrow{\alpha} E_2$, $R_1 \xrightarrow{\bar{\alpha}} R_2$ and $\alpha = a\langle \tilde{x} \rangle$, $\bar{\alpha} = (\nu \tilde{u})\bar{a}\langle \tilde{x} \rangle$. Moreover, let $(S_1, S_2) \in \mathcal{B}_1$ if $S_1 = (\nu \tilde{z})(E_1 \mid R_1^b)$ and $S_2 = (\nu \tilde{z} \tilde{u})(E_2 \mid R_2^b)$ where $E_1 \in \mathcal{E}^{\tilde{a}}$, $R_1 \in \mathcal{R}^{\tilde{a}}$, $E_1 \xrightarrow{\alpha} E_2$, $R_1 \xrightarrow{\tau} R_m \xrightarrow{\bar{\alpha}} R_2$, $\alpha = a\langle \tilde{x} \rangle$ and $\bar{\alpha} = (\nu \tilde{u})\bar{a}\langle \tilde{x} \rangle$. We show that $\mathcal{B}_0 \cup \mathcal{B}_1 \cup \simeq$ is a branching bisimulation.

First, suppose $(S_1, S_2) \in \mathcal{B}_0$ where $S_1 = (\nu \tilde{z})(E_1 \mid R_1^b)$, $S_2 = (\nu \tilde{z} \tilde{u})(E_2 \mid R_2^b)$ are as above. Suppose $S_2 \xrightarrow{\beta} Q_2$. Then $S_1 \xRightarrow{\beta} S'_1 = (\nu \tilde{z} \tilde{u})(E'_1 \mid R_1^b)$ where $E_1 \xRightarrow{\alpha} E'_1 \simeq E_2$ and $R_1 \xRightarrow{\bar{\alpha}} R'_1 \simeq R_2$. Since $R'_1 \simeq R_2$, $R_1^b \simeq R_2^b$. Hence $S'_1 \simeq S_2$ which implies that $S'_1 \xRightarrow{\beta} Q_1$ where $Q_1 \simeq Q_2$ as required. So suppose $S_1 \xrightarrow{\beta} Q_1$. The following possibilities exist:

1. $Q_1 \equiv (\nu \tilde{z} - \tilde{v})(E'_1 \mid R_1^b)$, $E_1 \xrightarrow{\beta} E'_1$ and β is τ , $x\langle \tilde{y} \rangle$ or $(\nu \tilde{w})\bar{x}\langle \tilde{y} \rangle$ where $\tilde{v} = \text{bn}(\beta)$. Since E_1 is A -confluent, by Lemma 4.1.5 we have that $E_2 \xRightarrow{\beta} E'_2$ and $E'_1 \xrightarrow{\alpha} E'_2$. Hence, as $\tilde{u} \cap x\tilde{y} = \emptyset$, $S_2 \xRightarrow{\beta} Q_2 = (\nu \tilde{z} \tilde{u} - \tilde{v})(E'_2 \mid R_2^b)$, where $(Q_1, Q_2) \in \mathcal{B}_0$.
2. $Q_1 \equiv (\nu \tilde{z})(E_1 \mid R_1^b)$, $R_1 \xrightarrow{\tau} R'_1$. By Lemma 4.3.5(ii), $R_1 \simeq R'_1$. So $R'_1 \xrightarrow{\bar{\alpha}} R_2$ and hence $(Q_1, S_2) \in \mathcal{B}_0$.
3. $Q_1 \equiv (\nu \tilde{z})(E_1 \mid R_1^b)$, $R_1 \xrightarrow{\beta} R'_1$, $\beta \neq \tau$. The arguments are similar to the first case.

4. $Q_1 \equiv (\nu\tilde{z}\tilde{w})(E'_1 \mid R_1^b)$, $E_1 \xrightarrow{\gamma} E'_1$, $R_1 \xrightarrow{\bar{\gamma}} R'_1$ where $\gamma \text{ comp } \bar{\gamma}$, $\tilde{w} = \text{bn}(\gamma) \cup \text{bn}(\bar{\gamma})$. Note that since $R_1 \in \mathcal{R}^a$, $\gamma \notin Q^-$. If $\gamma \in A^+$ then $\text{subj}(\gamma) = a$. So by A -confluence $\bar{\alpha} = \bar{\gamma}$, $E'_1 \simeq E_2$, $R'_1 \simeq R_2$ and $Q_1 \simeq S_2$. Otherwise, if $\gamma \notin A^+ \cup Q^-$ then the proof is a combination of arguments in Cases 1 and 3 above.

So suppose $(S_1, S_2) \in \mathcal{B}_1$. As before it is clear that if $S_2 \xrightarrow{\beta} Q_2$ then $S_1 \Rightarrow S'_1 = (\nu\tilde{z}\tilde{u})(E'_1 \mid R_1^b)$ where $E_1 \Rightarrow \xrightarrow{\alpha} E'_1 \simeq E_2$ and $R_1 \Rightarrow \xrightarrow{\tau} \xrightarrow{\bar{\alpha}} R'_1 \simeq R_2$. Hence $S'_1 \simeq S_2$. Moreover, $S'_1 \Rightarrow \xrightarrow{\beta} Q_1$ where $Q_1 \simeq Q_2$ as required. So suppose $S_1 \xrightarrow{\beta} Q_1$. The following possibilities exist:

1. $Q_1 \equiv (\nu\tilde{z} - \tilde{v})(E'_1 \mid R_1^b)$, $E_1 \xrightarrow{\beta} E'_1$ where β is τ , $x\langle\tilde{y}\rangle$ or $(\nu\tilde{w})\bar{x}\langle\tilde{y}\rangle$, and $\tilde{v} = \text{bn}(\beta)$. By Lemma 4.1.5, $E_2 \Rightarrow \xrightarrow{\beta} E'_2$, $E'_1 \xrightarrow{\alpha} \simeq E'_2$. Hence, as $\tilde{u} \cap x\tilde{y} = \emptyset$, $S_2 \Rightarrow \xrightarrow{\beta} Q_2 = (\nu\tilde{z}\tilde{u} - \tilde{v})(E'_2 \mid R_2^b)$, where $(Q_1, Q_2) \in \mathcal{B}_1$.
2. $Q_1 \equiv (\nu\tilde{z})(E_1 \mid R_1^b)$, $R_1 \xrightarrow{\tau} R'_1$. If $R_1 \simeq R'_1$ then $R'_1 \xrightarrow{\tau} \simeq \xrightarrow{\bar{\alpha}} \simeq R_2$ and so $(Q_1, S_2) \in \mathcal{B}_1$ as required. Otherwise, by Definition 4.3.4(5), $R'_1 \simeq R_m$. This implies that $R'_1 \xrightarrow{\bar{\alpha}} \simeq R_2$. Hence $(Q_1, S_2) \in \mathcal{B}_0$.
3. $Q_1 \equiv (\nu\tilde{z} - \tilde{v})(E_1 \mid R_1^b)$, $R_1 \xrightarrow{\beta} R'_1$, $\beta \neq \tau$ and β is $x\langle\tilde{y}\rangle$ or $(\nu\tilde{w})\bar{x}\langle\tilde{y}\rangle$ where $\tilde{v} = \text{bn}(\beta)$. Note that since $R_1 \xrightarrow{\tau} \simeq R_m \xrightarrow{\bar{\alpha}} \simeq R_2$ and $R_1 \in \mathcal{R}^a$ by Lemma 4.3.5(ii), R_1 is A^- -inert. Since $R'_1 \in \mathcal{R}^a$ then by Definition 4.3.4(2), either $R'_1 \Rightarrow \xrightarrow{\alpha'}$ or $R'_1 \Rightarrow \xrightarrow{\tau} \xrightarrow{\alpha'}$ where $\alpha' \in A^-$, $\text{subj}(\alpha') = a$. In the former case, since R_1 is Q, A -socially confluent, Definition 4.3.1(1) gives that $R_1 \Rightarrow \xrightarrow{\alpha'}$ which contradicts that R_1 is A^- -inert (note that $\beta \notin Q^\pm \cup A^\pm$ as all Q -names and all A -names are restricted). Hence it must be that

$$R_1 \xrightarrow{\beta} R'_1 \xrightarrow{\tau} \simeq R'_m \xrightarrow{\alpha'} \simeq R'_2.$$

Therefore by Definition 4.3.1(1) we have that

$$R_1 \xrightarrow{\tau} \simeq R_l \xrightarrow{\beta} \simeq R'_m \xrightarrow{\alpha'} \simeq R'_2$$

and by a further application of Definition 4.3.1(1)

$$R_1 \xrightarrow{\tau} \simeq R_l \xrightarrow{\alpha'} \simeq R'_l \xrightarrow{\beta} \simeq R'_2.$$

By Definition 4.3.4(5), $R_m \simeq R_l$ which by A -confluence implies that $\bar{\alpha} = \alpha'$ and thus $R_2 \simeq R'_l$. So $R_2 \xrightarrow{\beta} \simeq R'_2$. Since $R'_1 \xrightarrow{\tau} \simeq \xrightarrow{\bar{\alpha}} \simeq R'_2$, $(Q_1, (\nu\tilde{z}\tilde{u} - \tilde{v})(E_2 \mid R_2^b)) \in \mathcal{B}_1$ as required.

4. $Q_1 \equiv (\nu\tilde{z}\tilde{w})(E'_1 \mid R_1^b)$, $E_1 \xrightarrow{\gamma} E'_1$, $R_1 \xrightarrow{\bar{\gamma}} R'_1$, $\gamma \text{ comp } \bar{\gamma}$ and $\tilde{u} = \text{bn}(\gamma) \cup \text{bn}(\bar{\gamma})$. Note that since $R_1 \in \mathcal{R}^a$, $\gamma \notin Q^-$. Also, $\gamma \notin A^+$ as otherwise $\bar{\gamma} = \bar{\alpha}$ contradicting that R_1 is A^- -inert. The proof is then a combination of the arguments in the previous cases.

This completes the proof. \square

We now have the main theorem of this section which asserts that within a client context a server system is indistinguishable from its pruned version where at most one answer (A -action) may be outstanding at any point in time. The intuition underlying this result is the following: since

- for any question (Q -action) there exists at most one decisive action determining the answer, and
- each state-changing silent action determines an answer to a question,

determining the answer to a question may be thought of as an atomic action and therefore processing of questions can be serialized. So let P be the original system where questions may be processed concurrently, and S the variation of it with a serial server. Clearly, any computation of S is also (essentially) a computation of P . However, the converse does not hold: there are states of P , where more than one question is being concurrently processed by the server component, that are not mirrored by S . Nonetheless, S may simulate P up to branching bisimilarity. A key observation in seeing that S can ‘match’ any computation of P is that the server of S may postpone accepting a question which has been accepted by the server of P until the decisive action (if it exists) is performed. The serial system may then accept and process the question and return the result. However, since the answer to the question has been determined in P by the decisive silent action, by A -confluence no difference in behaviour is observable. These intuitions are formalized in the proof.

Theorem 4.3.8 Suppose λ is a Q, A -sorting. Further suppose \mathcal{R} is a (Q^+, A^-) -server with a (Q^+, A^-) -server partition $\{\mathcal{R}^{\tilde{a}}\}_{\tilde{a}}$ and \mathcal{E} is a (Q^-, A^+) -client with (Q^-, A^+) -client partition $\{\mathcal{E}^{\tilde{a}}\}_{\tilde{a}}$. Suppose $R \in \mathcal{R}^{\emptyset}$, $E \in \mathcal{E}^{\emptyset}$ and $(\nu \tilde{z})(E \mid R)$ is Q, A -closed. Then $(\nu \tilde{z})(E \mid R) \dot{\simeq} (\nu \tilde{z})(E \mid R^b)$.

PROOF: Let $(S_1, S_2) \in \mathcal{B}$ if $S_1 = (\nu \tilde{z})(E_1 \mid R_1)$ and $S_2 = (\nu \tilde{z}')(E_2 \mid R_2^b)$ where $E_2 \in \mathcal{E}^{\emptyset}$, $R_2 \in \mathcal{R}^{\emptyset}$ and there exist E, R, s, u, \tilde{a} , such that $R \in \mathcal{R}^{\tilde{a}}$, $E \in \mathcal{E}^{\tilde{a}}$, R is A^- -inert and

$$\begin{aligned} E_1 &\xrightarrow{s} \dot{\simeq} E, & R_1 &\xrightarrow{\tilde{s}} \dot{\simeq} R, & \tilde{s} &= \overline{\alpha_1} \dots \overline{\alpha_n}, & \overline{\alpha_i} &= (\nu \tilde{p}_i) \overline{r_i} \langle \tilde{v}_i \rangle, & s &\text{comp } \tilde{s} \\ E_2 &\xrightarrow{\tilde{u}} \dot{\simeq} E, & R_2 &\xrightarrow{u} \dot{\simeq} R, & \tilde{u} &= \overline{\beta_1} \dots \overline{\beta_k}, & \overline{\beta_i} &= (\nu \tilde{y}_i) \overline{q_i} \langle \tilde{x}, b_i \rangle, & u &\text{comp } \tilde{u}, \end{aligned}$$

where $r_i, b_i : A$, $r_1 \dots r_n$ are pairwise distinct, $q_i : Q$, $b_i \in \tilde{y}_i$, and $\tilde{u} = \{b_1, \dots, b_k\}$. We show that $\mathcal{B} \dot{\simeq} \cup \dot{\simeq}$ is a branching bisimilarity.

First suppose $(S_1, S_2) \in \mathcal{B}$ where $S_1 = (\nu \tilde{z})(E_1 \mid R_1)$ and $S_2 = (\nu \tilde{z}')(E_2 \mid R_2^b)$ are as above. Suppose $S_2 \xrightarrow{\rho} Q_2$. We show that $S_1 \Rightarrow S'_1 \xrightarrow{\rho} Q_1$ where $(Q_1, Q_2) \in \mathcal{B} \dot{\simeq}$ and $(S'_1, S_2) \in \mathcal{B} \dot{\simeq}$. The following cases exist:

1. $Q_2 \equiv (\nu\tilde{z}' - \tilde{v})(E'_2 \mid R_2^b)$ and $E_2 \xrightarrow{\rho} E'_2$ where ρ is τ , $x(\tilde{y})$ or $(\nu\tilde{w})\bar{x}(\tilde{y})$ with $\tilde{v} = \text{bn}(\rho)$. Then since $E_2 \xrightarrow{\bar{u}} \simeq E$ and E_2 is Q -confluent, by Lemma 4.1.5, $E \Rightarrow \xrightarrow{\rho} E'$ and $E_2 \xrightarrow{\bar{u}} \simeq E'$. In turn, $S_1 \Rightarrow S'_1 = (\nu\tilde{z}\tilde{p})(E'_1 \mid R'_1)$, where $E_1 \xrightarrow{s} \simeq E'_1 \simeq E$, $R_1 \xrightarrow{\bar{s}} \simeq R'_1 \simeq R$ and $\tilde{p} = \tilde{p}_1 \dots \tilde{p}_n$. Clearly, $(S'_1, S_2) \in \mathcal{B}$. In addition, $E'_1 \Rightarrow \xrightarrow{\rho} E''_1 \simeq E'$ and $((\nu\tilde{z}\tilde{p} - \tilde{v})(E''_1 \mid R'_1), Q_2) \in \mathcal{B}$ as required.
2. $Q_2 \equiv (\nu\tilde{z}' - \tilde{w})(E_2 \mid R_2^b)$ and $R_2 \xrightarrow{\rho} R'_2$, $\text{bn}(\rho) = \tilde{w}$. Since $R_2 \in \mathcal{R}^\emptyset$, if $\rho = \tau$ then by Lemma 4.3.5(i), $R_2 \simeq R'_2$. Under this condition, the proof follows similarly to the previous case.
3. $\rho = \tau$ and $Q_2 \equiv (\nu\tilde{z}')(\bar{E}'_2 \mid R_2^b)$ as $E_2 \xrightarrow{\gamma} E'_2$, $R_2 \xrightarrow{\bar{\gamma}} R'_2$ where $\gamma \text{ comp } \bar{\gamma}$ and $\gamma \notin Q^-$. Since $R_2 \in \mathcal{R}^\emptyset$, $\gamma \notin A^+$. The proof is a combination of the two previous cases. (Note that it is now also possible that $\gamma \in A^- \cup Q^+$.)
4. $\rho = \tau$ and $Q_2 \equiv (\nu\tilde{z}'\tilde{x})(E'_2 \mid R_2^b)$ as $E_2 \xrightarrow{\bar{\rho}} E'_2$, $R_2 \xrightarrow{\rho} R'_2$ where $\rho = q(\tilde{y}, a)$, $\bar{\rho} = (\nu\tilde{x})\bar{q}(\tilde{y}, a)$, $q : Q$, $a \in \tilde{x}$ and $a : A$. Clearly, $S_1 \Rightarrow S'_1 = (\nu\tilde{z}\tilde{p})(E'_1 \mid R'_1)$, where $E_1 \xrightarrow{s} \simeq E'_1 \simeq E$, $R_1 \xrightarrow{\bar{s}} \simeq R'_1 \simeq R$ and $\tilde{p} = \tilde{p}_1 \dots \tilde{p}_n$. Moreover, $(S'_1, S_2) \in \mathcal{B}$. There are two cases depending on whether $\rho = \beta_i$ for some i .

- Suppose $\rho = \beta_i$ and $\rho \neq \beta_j$ for all $j < i$. Then by Definition 4.3.1(1), $R_2 \xrightarrow{\rho} \simeq \xrightarrow{u'} \simeq R$ and $E_2 \xrightarrow{\bar{\rho}} \simeq \xrightarrow{\bar{u}'} \simeq E$ where $u' = u/\rho$, $u' \text{ comp } \bar{u}'$. Further, by Lemma 4.1.5, $E'_2 \xrightarrow{\bar{u}'} \simeq E$ and $R'_2 \xrightarrow{u'} \simeq R$. Moreover, $E'_2 \in \mathcal{E}^a$ and $R'_2 \in \mathcal{R}^a$ so by Definition 4.3.6(6) and Definition 4.3.4(2) respectively, we have that $E'_2 \xrightarrow{\alpha} E''_2$ and either $R'_2 \xrightarrow{\bar{\alpha}} \simeq R''_2$ or $R'_2 \xrightarrow{\tau} \simeq \xrightarrow{\bar{\alpha}} \simeq R''_2$ where $\alpha \text{ comp } \bar{\alpha}$ and $\text{subj}(\alpha) = a$. Suppose $R'_2 \xrightarrow{\bar{\alpha}} \simeq R''_2$: then by Lemma 4.1.5, $R \xrightarrow{\bar{\alpha}} \simeq R'$. This contradicts the construction of \mathcal{B} , in particular the assumption of R being A^- -inert. Hence it must be that $R'_2 \xrightarrow{\tau} \simeq \xrightarrow{\bar{\alpha}} \simeq R''_2$. So since $E'_2 \xrightarrow{\alpha} E''_2$ and $R'_2 \xrightarrow{\tau} \simeq \xrightarrow{\bar{\alpha}} \simeq R''_2$, by Lemma 4.1.5, we have

$$E \xrightarrow{\alpha} \simeq E', \quad R \xrightarrow{\tau} \simeq \xrightarrow{\bar{\alpha}} \simeq R'$$

where

$$E''_2 \xrightarrow{\bar{u}'} \simeq E', \quad R''_2 \xrightarrow{u'} \simeq R'.$$

Since $E'_1 \simeq E$ and $R'_1 \simeq R$,

$$E'_1 \Rightarrow \xrightarrow{\alpha} E''_1 \simeq E'$$

and

$$R'_1 \Rightarrow \xrightarrow{\tau} \Rightarrow \xrightarrow{\bar{\alpha}} R''_1 \simeq R'.$$

Hence, $S'_1 \Rightarrow Q_1 = (\nu \tilde{z} \tilde{p})(E''_1 \mid R''_1)$, where $(Q_1, (\nu \tilde{z}' \tilde{x})(E''_2 \mid R''^b_2)) \in \mathcal{B}$. By Theorem 4.3.7, $Q_2 \simeq (\nu \tilde{z}' \tilde{x})(E''_2 \mid R''^b_2)$ so $(Q_1, Q_2) \in \mathcal{B} \simeq$.

- Now suppose $\rho \notin u$. By Q -confluence and Lemma 4.1.5, $E \Rightarrow \xrightarrow{\bar{\rho}} E'$, where $E'_2 \xrightarrow{\bar{u}} \simeq E$. Moreover, since $R_2 \xrightarrow{\rho} R'_2$ and $R_2 \xrightarrow{u} \simeq R$, by Definition 4.3.1(2) we have that $R \Rightarrow \xrightarrow{\rho} R'$ where by repeated application of Definition 4.3.1(1), $R'_2 \xrightarrow{u} \simeq R'$. Hence $E'_1 \Rightarrow \xrightarrow{\bar{\rho}} E''_1 \simeq E'$, $R'_1 \Rightarrow \xrightarrow{\rho} \simeq R'_1 \simeq R'$ and $S'_1 \Rightarrow S''_1 = (\nu \tilde{z} \tilde{p})(E''_1 \mid R''_1)$. Since $E'_2 \in \mathcal{E}^a$ and $R'_2 \in \mathcal{R}^a$, $E_2 \xrightarrow{\alpha} E''_2$ and $R'_2 \xrightarrow{\bar{\alpha}} R''_2$, where $\alpha \text{ comp } \bar{\alpha}$, $\text{subj}(\alpha) = a$. Using arguments similar to those in the previous case we can show that $S''_1 \Rightarrow Q_1$, where $(Q_1, Q_2) \in \mathcal{B} \simeq$.

Now suppose $(S_1, S_2) \in \mathcal{B}$ and $S_1 \xrightarrow{\rho} Q_1$. There are several cases.

1. Suppose $Q_1 \equiv (\nu \tilde{z} - \tilde{v})(E'_1 \mid R_1)$, where $E_1 \xrightarrow{\rho} E'_1$ and ρ is $\tau, x\langle \tilde{y} \rangle$ or $(\nu \tilde{w})\bar{x}\langle \tilde{y} \rangle$ with $\tilde{v} = \text{bn}(\rho)$. If $E_1 \simeq E'_1$ then as $E'_1 \xrightarrow{\rho} \simeq E$, $(Q_1, S_2) \in \mathcal{B}$ as required. Otherwise, since E_1 is A -confluent, by Lemma 4.1.5, there is E' such that $E \Rightarrow \xrightarrow{\rho} E'$ and $E'_1 \xrightarrow{\rho} \simeq E'$. Since $E_2 \in \mathcal{E}^0$ and $E_2 \xrightarrow{\bar{u}} \simeq E \Rightarrow \xrightarrow{\rho} E'$, by repeated application of (1) in Definition 4.3.1, $E_2 \Rightarrow \xrightarrow{\rho} E'_2$ and $E'_2 \xrightarrow{\bar{u}} \simeq E'$. Hence $S_2 \Rightarrow \xrightarrow{\rho} Q_2 = (\nu \tilde{z}' - \tilde{v})(E'_2 \mid R'_2)$ and $(Q_1, Q_2) \in \mathcal{B}$.
2. Suppose $Q_1 \equiv (\nu \tilde{z} - \tilde{v})(E_1 \mid R'_1)$, where $R_1 \xrightarrow{\rho} R'_1$ and ρ is $x\langle \tilde{y} \rangle$ or $(\nu \tilde{z})\bar{x}\langle \tilde{y} \rangle$ with $\tilde{v} = \text{bn}(\rho)$. Note that $\rho \notin A^\pm$ as all A -names are restricted. So by Lemma 4.1.5 there is R' such that $R \Rightarrow \xrightarrow{\rho} R'$ and $R'_1 \xrightarrow{\bar{s}} \simeq R'$. Since $R_2 \in \mathcal{R}^0$ and $R_2 \xrightarrow{u} \simeq R \Rightarrow \xrightarrow{\rho} R'$, by repeated application of Definition 4.3.1(1), $R_2 \Rightarrow \xrightarrow{\rho} R'_2$ and $R'_2 \xrightarrow{u} \simeq R'$. Hence $S_2 \Rightarrow \xrightarrow{\rho} Q_2 = (\nu \tilde{z}' - \tilde{v})(E_2 \mid R'_2)$ and $(Q_1, Q_2) \in \mathcal{B}$.
3. Suppose $Q_1 \equiv (\nu \tilde{z})(E_1 \mid R'_1)$, where $R_1 \xrightarrow{\tau} R'_1$. If $R_1 \simeq R'_1$ then $R'_1 \xrightarrow{\bar{s}} \simeq R$ and hence $(Q_1, S_2) \in \mathcal{R}$ as required. Otherwise, we have the following: since $R_1 \xrightarrow{\bar{s}} \simeq R$, by Q -confluence and Lemma 4.1.5,

$$R \Rightarrow \xrightarrow{\tau} R' \text{ and } R'_1 \xrightarrow{\bar{s}} \simeq R'.$$

Further since R is A^- -inert, by Definition 4.3.4(3), $R' \Rightarrow \xrightarrow{\alpha} R''$, where $\alpha \in A^-$. Thus by Definition 4.3.1(1), $R'_1 \Rightarrow \xrightarrow{\alpha} R''_1$ and $R''_1 \xrightarrow{\bar{s}} \simeq R''$. Note that by Definition 4.3.4(3), R'' is A^- -inert.

Recall that $R_2 \in \mathcal{R}^0$ and $R_2 \xrightarrow{u} \simeq R$, where $u \in Q^{++}$. Then by Definition 4.3.4(4),

$$R_2 \xrightarrow{\beta} \simeq \xrightarrow{\tau} \simeq R'_2 \text{ where } R'_2 \xrightarrow{u/\beta} \simeq R'$$

for $\beta \in u$. By Lemma 4.3.5(ii) and Definition 4.3.4(3), there is $\alpha' \in A^-$ such that $R'_2 \Rightarrow^{\alpha'} R''_2$. By Q -confluence and Lemma 4.1.5, we have that $R' \Rightarrow^{\alpha'} T$. Two cases exist: either $\text{subj}(\alpha') = \text{subj}(\alpha)$ and by A -confluence $\alpha = \alpha'$, $R'' \simeq T$, or, $\text{subj}(\alpha) \neq \text{subj}(\alpha')$ and by A -confluence $R'' \Rightarrow^{\alpha'}$. However, since R'' is A^- -inert this is a contradiction. Thus $\alpha' = \alpha$ and $R''_2 \xrightarrow{u/\beta} \simeq R''$.

Further, since $u \text{ comp } \bar{u}$, there is $\bar{\beta} \text{ comp } \beta$ such that $\bar{\beta} \in \bar{u}$ and as $E_2 \xrightarrow{\bar{u}} \simeq E$, by Q , A -social confluence, $E_2 \xrightarrow{\bar{\beta}} \simeq E'_2 \xrightarrow{\bar{u}/\bar{\beta}} \simeq E$, where $E'_2 \in \mathcal{E}^a$. Hence by Definition 4.3.6(6), $E'_2 \xrightarrow{\bar{\alpha}} E''_2$, where $\alpha \text{ comp } \bar{\alpha}$ and by Lemma 4.1.5, $E''_2 \xrightarrow{\bar{u}/\bar{\beta}} \simeq E'$ and $E \xrightarrow{\bar{\alpha}} \simeq E'$. So, $S_2 \Rightarrow^{\tau} Q_2 = (\nu \tilde{z}' \tilde{p})(E'_2 \mid R''_2)$ for $\tilde{p} = \text{bn}(\alpha)$, and since $E_1 \xrightarrow{s\bar{\alpha}} \simeq E'$, $R'_1 \xrightarrow{s\alpha} \simeq R''$, $(Q_1, Q_2) \in \mathcal{B}$ as required.

4. If $\alpha = \tau$ and $Q_1 = (\nu \tilde{z})(E'_1 \mid R'_1)$ where $E_1 \xrightarrow{\gamma} E'_1$, $R_1 \xrightarrow{\bar{\gamma}} R'_1$ and $\gamma \notin Q^- \cup A^+$ then we combine elements of the arguments in the first two cases above.
5. Suppose $\alpha = \tau$ and $Q_1 = (\nu \tilde{z}' \tilde{w})(E'_1 \mid R'_1)$ where $E_1 \xrightarrow{\bar{\gamma}} E'_1$, $R_1 \xrightarrow{\gamma} R'_1$, $\gamma = q\langle \tilde{x}, a \rangle$, $\bar{\gamma} = (\nu \tilde{w})\bar{q}\langle \tilde{x}, a \rangle$, $q : Q$ and $a \in \tilde{w}$. By Q -confluence and Lemma 4.1.5, $E' \xrightarrow{\bar{\gamma}} \simeq E'$, $R \xrightarrow{\gamma} \simeq R'$ where $E'_1 \xrightarrow{s} \simeq E'$ and $R'_1 \xrightarrow{s} \simeq R'$. There are two cases to consider:

- Suppose R' is A^- -inert. Clearly, $E_2 \xrightarrow{u\bar{\gamma}} \simeq E'$ and $R_2 \xrightarrow{u\gamma} \simeq R'$. So $(Q_1, S_2) \in \mathcal{B}$.
- On the other hand, if $R' \xrightarrow{\alpha} R''$ for some $\alpha \in A^-$ then $R'_1 \xrightarrow{s\alpha} \simeq R''$. Note that by Lemma 4.3.5(iii), R'' is A^- -inert. Recall that $R_2 \in \mathcal{R}^\emptyset$, $R_2 \xrightarrow{u} \simeq R \xrightarrow{\gamma} \simeq R' \xrightarrow{\alpha} R''$ where $u \in Q^{+*}$. Then by Definition 4.3.4(4),

$$R_2 \xrightarrow{\beta} \simeq R'_2 \xrightarrow{\alpha} \simeq R''_2, \text{ where } R'_2 \xrightarrow{u\gamma/\beta} \simeq R''$$

for some $\beta \in u\gamma$. Suppose $\beta \in u$. Then by Q -confluence and Lemma 4.1.5, $R \Rightarrow^{\alpha}$ which contradicts the A^- -inertness of R . Thus $\beta = \gamma$. Moreover, since $E_2 \xrightarrow{\bar{u}} \simeq E \xrightarrow{\bar{\gamma}} \simeq E'$, by Definition 4.3.1(1), $E_2 \xrightarrow{\bar{\gamma}} \simeq E'_2$ where $E'_2 \xrightarrow{\bar{u}} \simeq E'$. Further, by Definition 4.3.6(6), $E'_2 \xrightarrow{\bar{\alpha}} E''_2$, where $\alpha \text{ comp } \bar{\alpha}$ and by A -confluence and Lemma 4.1.5, $E' \Rightarrow^{\bar{\alpha}} E''$ where $E''_2 \xrightarrow{\bar{u}} \simeq E''$. So, with $\tilde{x} = \text{bn}(\alpha)$,

$$S_2 \Rightarrow S'_2 = (\nu \tilde{z}' \tilde{w})(E'_2 \mid R'_2) \Rightarrow Q_2 = (\nu \tilde{z}' \tilde{w} \tilde{x})(E''_2 \mid R''_2)$$

where $R'_2 \xrightarrow{u} \simeq R''$ and $E''_2 \xrightarrow{\bar{u}} \simeq E''$. Since also $R'_1 \xrightarrow{s\alpha} \simeq R''$ and $E_1 \xrightarrow{s\bar{\alpha}} \simeq E''$, $(Q_1, Q_2) \in \mathcal{B}$.

6. Finally, suppose $\alpha = \tau$ and $Q_1 = (\nu \tilde{z} \tilde{x})(E'_1 \mid R'_1)$ where $E_1 \xrightarrow{\gamma} E'_1$, $R_1 \xrightarrow{\bar{\gamma}} R'_1$ and $\gamma = a\langle \tilde{y} \rangle$, $\bar{\gamma} = (\nu \tilde{x})\bar{a}\langle \tilde{y} \rangle$, $a : A$. Then $\gamma \in s$ as otherwise, by A -confluence and Lemma 4.1.5, $R \xrightarrow{\bar{\gamma}}_{\simeq} R'$ which contradicts the construction of \mathcal{B} and in particular the fact that R is A^- -inert. So by A -confluence and Lemma 4.1.5, $R'_1 \xrightarrow{\bar{\gamma}/\bar{\gamma}}_{\simeq} R$. Since additionally, by the construction of \mathcal{R} , $\text{subj}(\bar{\gamma}) \notin \text{subj}(\bar{s}/\bar{\gamma})$ then by A -confluence and Lemma 4.1.5, $E'_1 \xrightarrow{s/\gamma}_{\simeq} E$. Thus $(Q_1, S_2) \in \mathcal{B}$.

We have shown that if $(P, Q) \in \mathcal{B}$ and $P \xrightarrow{\alpha} P'$ then $Q \Rightarrow Q'' \xrightarrow{\alpha} Q'$ where $(P, Q''), (P', Q') \in \mathcal{B} \simeq$ and vice versa. We may now complete the proof of $\mathcal{B} \simeq \subseteq \simeq$. Suppose $(S_1, S_2) \in \mathcal{B} \simeq$ and $S_1 \xrightarrow{\alpha} Q_1$. Suppose S is such that $(S_1, S) \in \mathcal{B}$ and $S \simeq S_2$. By our previous result, $S \Rightarrow S' \xrightarrow{\alpha} Q$ where $(S_1, S'), (Q_1, Q) \in \mathcal{B} \simeq$. Moreover, since $S \simeq S_2$, $S_2 \Rightarrow \xrightarrow{\alpha} Q_2 \simeq Q$ and $(Q_1, Q_2) \in \mathcal{B} \simeq$ as required. On the other hand, if $S_2 \xrightarrow{\alpha} Q_2$ then $S \Rightarrow \xrightarrow{\alpha} Q$ where $Q_2 \simeq Q$. Since $(S_1, S) \in \mathcal{B}$, $S_1 \Rightarrow \xrightarrow{\alpha} Q_1$, where $(Q_1, Q) \in \mathcal{B} \simeq$. Thus $(Q_1, Q_2) \in \mathcal{B} \simeq$ which completes the proof. \square

We may obtain the following special case of Theorem 4.3.8 by restricting attention to A -polite agents.

Theorem 4.3.9 Suppose λ is a Q, A -sorting. Further, suppose that E is A -polite and $(\nu \tilde{z})(E \mid R)$ is a Q, A -closed agent. Let \mathcal{E} be the transition system generated by E and \mathcal{R} the transition system generated by R and suppose that \mathcal{R} is a (Q^+, A^-) -server and \mathcal{E}^A is a (Q^-, A^+) -client. Then $(\nu \tilde{z})(E \mid R) \simeq (\nu \tilde{z})(E \mid R^b)$.

PROOF: The proof is a simple corollary of Theorem 4.3.8. \square

4.4 When names are not new

Recall the definition of an (M^-, R^+) -ready (resp. (M^+, R^-) -disciplined) partition. As we have already pointed out, the definition imposes requirements only on transitions of agents of the partition, involving actions $\alpha \in M^-$ (resp. M^+) where $\text{obj}_R(\alpha)$ is a new name. That is, during the interaction of a ready agent and a disciplined agent it is necessary that, on asking a question, the ready system provides a new name via which the answer will be returned. It may be argued that this is a reasonable condition to impose. For suppose the ready component of a system asks two questions of the disciplined component and requires the answers to be returned via the same name. It appears that when the answers are returned, confusion may arise as to which answer corresponds to what question. However, recall that a ready system is R -confluent. This suggests that possibly this confusion is not as serious as it might seem. For let $S \stackrel{\text{def}}{=} r(x).P \mid r(x).Q$ be a ready system, awaiting along

name $r : R$ for the answers to two questions. It may engage in the following two computations with visible content $r\langle v \rangle r\langle v' \rangle$:

$$\begin{aligned} S &\xrightarrow{r\langle v \rangle} S'_1 = P\{v/x\} \mid r(x).Q & \xrightarrow{r\langle v' \rangle} S_1 = P\{v/x\} \mid Q\{v'/x\} \\ S &\xrightarrow{r\langle v \rangle} S'_2 = r(x).P \mid Q\{v/x\} & \xrightarrow{r\langle v' \rangle} S_2 = P\{v'/x\} \mid Q\{v/x\} \end{aligned}$$

By R -confluence we have that $S'_1 \simeq S'_2$ and thus $S'_2 \xRightarrow{r\langle v' \rangle} S''_2 \simeq S_1$. Since, by R -confluence, S'_2 may engage in an input via r in exactly one way, $S''_2 \simeq S_2$ and so $S_1 \simeq S_2$. So, although the answers to the questions may be received in various ways, they all lead to the same state up to branching bisimilarity.

In this section we investigate how the theory can be modified to relax the condition on the uniqueness of R -names. Although we work with the notion of branching bisimilarity and ignore divergence, it is believed that the results may be extended to the divergence-sensitive setting. We begin with the notion of R -confluence. According to the original definition, an R -confluent agent is capable of performing a unique output via an R -action. However, since we want to consider agents that may use an R -name for output in more than one ways, we modify the definition as follows:

Definition 4.4.1 Let R be a set of sorts. A process P is R^δ -confluent if for every derivative Q of P the following hold:

1. if $\rho \in R^\pm$, $\text{subj}(\alpha) \neq \text{subj}(\rho)$, $Q \xrightarrow{\rho} Q_1$ and $Q \Rightarrow \xrightarrow{\alpha} Q_2$, then for some Q' , $Q_1 \Rightarrow \xrightarrow{\alpha} Q'$ and $Q_2 \Rightarrow \xrightarrow{\rho} \simeq Q'$;
2. if $\rho_1, \rho_2 \in R^-$, $\text{subj}(\rho_1) = \text{subj}(\rho_2)$, $Q \xrightarrow{\rho_1} Q_1$ and $Q \Rightarrow \xrightarrow{\rho_2} Q_2$, then either
 - (a) $\rho_1 = \rho_2$ and $Q_1 \simeq Q_2$, or
 - (b) for some Q' , $Q_1 \Rightarrow \xrightarrow{\rho_2} Q'$ and $Q_2 \Rightarrow \xrightarrow{\rho_1} \simeq Q'$;
3. if $\rho \in R^+$, $Q \xrightarrow{\rho} Q_1$ and $Q \Rightarrow \xrightarrow{\rho} Q_2$, then $Q_1 \simeq Q_2$.

According to the definition,

$$A \stackrel{\text{def}}{=} \bar{r}\langle x \rangle. \bar{a}. 0 \mid \bar{r}\langle x \rangle. \bar{b}. 0$$

$$B \stackrel{\text{def}}{=} \bar{r}\langle x \rangle. \bar{a}. 0 \mid \bar{r}\langle y \rangle. \bar{b}. 0$$

are R^δ -confluent agents. This shows that an R^δ -confluent agent is not, in general, determinate with respect to R -actions: A has two $\bar{r}\langle x \rangle$ -derivatives, which are not equivalent. However, it delays determinacy (hence the δ in R^δ -confluence) as specified by Clause 2(b). Moreover, an R^δ -confluent agent, unlike an R -confluent one, may perform distinct R^- -actions as demonstrated by agent B . R^δ -confluence is preserved by \simeq .

Lemma 4.4.2 If P is R^δ -confluent and $P \dot{\simeq} Q$, then Q is R^δ -confluent.

PROOF: Let Q_0 be a derivative of Q . Then since $P \dot{\simeq} Q$ there is a derivative P_0 of P such that $P_0 \dot{\simeq} Q_0$.

1. Suppose $Q_0 \xrightarrow{\rho} Q_1, Q_0 \Rightarrow^\alpha Q_2$ where $\rho \in R^\pm$ and $\text{subj}(\alpha) \neq \text{subj}(\rho)$. Since $Q_0 \xrightarrow{\rho} Q_1, P_0 \Rightarrow P'_0 \xrightarrow{\rho} P_1$ for some P'_0, P_1 with $P'_0 \dot{\simeq} Q_0$ and $P_1 \dot{\simeq} Q_1$. Then since $P'_0 \dot{\simeq} Q_0$ and $Q_0 \Rightarrow^\alpha Q_2, P'_0 \Rightarrow^\alpha P_2$ for some P_2 with $P_2 \dot{\simeq} Q_2$. Being a derivative of P , P'_0 is R^δ -confluent. Hence, there is P' such that $P_1 \Rightarrow^\alpha P'$ and $P_2 \xrightarrow{\rho} P'$, and so there is Q' as required.
2. Suppose $Q_0 \xrightarrow{\rho_1} Q_1$ and $Q_0 \Rightarrow Q'_2 \xrightarrow{\rho_2} Q_2$ where $\rho_1, \rho_2 \in R^-$ and $\text{subj}(\rho_1) = \text{subj}(\rho_2)$. Then $P_0 \Rightarrow P'_0 \xrightarrow{\rho_1} P_1$ with $P'_0 \dot{\simeq} Q_0$ and $P_1 \dot{\simeq} Q_1$, and as $P'_0 \dot{\simeq} Q_0, P'_0 \Rightarrow P'_2 \xrightarrow{\rho_2} P_2$ with $P'_2 \dot{\simeq} Q_2$. Since P'_0 is R^δ -confluent, either $\rho_1 = \rho_2$ and $P_1 \dot{\simeq} P_2$ and so $Q_1 \dot{\simeq} Q_2$, or there exists P' such that $P_1 \Rightarrow P'$ and $P_2 \xrightarrow{\rho_1} P'$. Thus there exists Q' such that $Q_1 \Rightarrow Q'$ and $Q_2 \xrightarrow{\rho_1} Q'$.
3. Suppose $Q_0 \xrightarrow{\rho} Q_1$ and $Q_0 \Rightarrow Q'_2 \xrightarrow{\rho} Q_2$, where $\rho \in R^+$. Then $P_0 \Rightarrow P'_0 \xrightarrow{\rho} P_1$ with $P'_0 \dot{\simeq} Q_0$ and $P_1 \dot{\simeq} Q_1$, and as $P'_0 \dot{\simeq} Q_0, P'_0 \Rightarrow P'_2 \xrightarrow{\rho} P_2$ with $P'_2 \dot{\simeq} Q_2$. Since P'_0 is R^δ -confluent, $P_1 \dot{\simeq} P_2$ and so $Q_1 \dot{\simeq} Q_2$. \square

However, the notion of R^δ -confluence does not satisfy the Lemma 4.1.5. For example, consider agent A given above. It may engage in the following three transitions, where $s = \bar{r}\langle x \rangle a \bar{r}\langle x \rangle$:

$$\begin{aligned} A &\xrightarrow{s} A_1 = b.0 \\ A &\xrightarrow{\bar{r}\langle x \rangle} A_2 = a \mid \bar{r}\langle x \rangle. b.0 \\ A &\xrightarrow{\bar{r}\langle x \rangle} A_3 = \bar{r}\langle x \rangle. a \mid b.0 \end{aligned}$$

Although $A_2 \xrightarrow{s/\bar{r}\langle x \rangle} A_1, A_3 \not\xrightarrow{a}$. To obtain a variant of Lemma 4.1.5 we introduce the following definition.

Definition 4.4.3 $s' \in s - \alpha$ iff for some $s_1, s_2, s = s_1 \alpha s_2, s' = s_1 s_2$.

Thus $s - \alpha$ denotes the set of all subsequences of s with an occurrence of α removed. Clearly, in the example, there exists $s' \in s - \bar{r}\langle x \rangle$ such that $A_3 \xrightarrow{s'} A_1$. The result formalizing this property of R^δ -confluent agents follows:

Lemma 4.4.4 Suppose P is R^δ -confluent, $s \in R^{\pm*}, P \xrightarrow{s} P_1$ and $P \Rightarrow^\alpha P_2$, where if $\alpha \in R^+$ and $s = s_0 \rho s_1$, with $\text{subj}(\rho) = \text{subj}(\alpha)$ and $\text{subj}(\alpha) \notin \text{subj}(s_0)$, then $\alpha = \rho$. Then, if $\alpha \in s$ and $\alpha \in R^+, P_2 \xrightarrow{s/\alpha} P_1$, otherwise, either

- $\alpha \in s$, $\alpha \in R^-$ and $P_2 \xrightarrow{s'} \simeq P_1$, for some $s' \in s - \alpha$, or
- there exists P_0 such that $P_1 \Rightarrow \xrightarrow{\alpha} P_0$ and $P_2 \xrightarrow{s} \simeq P_0$.

PROOF: By induction on s . If s is the empty sequence then $\alpha \notin s$ and by definition of \simeq , $P_1 \Rightarrow \xrightarrow{\alpha} P_0$ with $P_2 \simeq P_0$. So suppose $s = t\rho$ and $P \xrightarrow{t} \simeq P'_1 \xrightarrow{\rho} \simeq P_1$.

First suppose $\alpha \in s$, $\alpha \in R^+$. If $\alpha \in t$ then $P_2 \xrightarrow{t/\alpha} \simeq P'_1$ and so $P_2 \xrightarrow{s/\alpha} \simeq P_1$. Further, if $\alpha \notin t$ then $\text{subj}(\alpha) \notin \text{subj}(t)$ and $\alpha = \rho$. Moreover, by the induction hypothesis, $P'_1 \Rightarrow \xrightarrow{\alpha} P'_0$, $P_2 \xrightarrow{t} \simeq P'_0$. By R^δ -confluence $P'_0 \simeq P_1$ so $P_2 \xrightarrow{s/\alpha} \simeq P_1$.

Otherwise by the induction hypothesis two possibilities exist.

- $\alpha \in s$, $\alpha \in R^-$ and $P_2 \xrightarrow{t'} \simeq P'_1$, where $t' \in t - \alpha$. Then $P_2 \xrightarrow{t'\rho} \simeq P_1$ and $t'\rho \in s - \alpha$ as required.
- $P'_1 \Rightarrow \xrightarrow{\alpha} P'_0$ and $P_2 \xrightarrow{t} \simeq P'_0$. If $\alpha = \rho$ then by R^δ -confluence, either $P'_0 \simeq P_1$, in which case $P_2 \xrightarrow{t} \simeq P_1$ and $t \in s - \alpha$, or $P_1 \Rightarrow \xrightarrow{\alpha} P_0$ and $P'_0 \xrightarrow{\alpha} \simeq P_0$ for some P'_0 . So $P_2 \xrightarrow{s} \simeq P_0$, $P_1 \Rightarrow \xrightarrow{\alpha} P_0$; otherwise, if $\alpha \notin s$ then by R^δ -confluence, there exists P_0 such that $P_1 \Rightarrow \xrightarrow{\alpha} P_0$, $P'_0 \xrightarrow{\rho} \simeq P_0$ and so $P_2 \xrightarrow{s} \simeq P_0$, $P_1 \Rightarrow \xrightarrow{\alpha} P_0$. \square

We continue with the modifications of tidy, ready and disciplined partitions. First note that it is now necessary to have multiset indices to blocks of the partition as opposed to sets since names may be used to answer more than one question. In addition our new definitions refine the restrictions imposed on a ready partition. The purpose of this is to exclude derivatives of the form $r(x).r(y).\bar{a}(y)$ where the order in which two inputs via name r are received can be distinguished. The significance of this will become apparent in the proof.

Definition 4.4.5 Suppose λ is an M, R -sorting. A derivation-closed set S of R^δ -confluent processes is $(M^-, R^+)^\delta$ -tidy if there is a partition $\{\mathcal{S}^{\tilde{r}} \mid \tilde{r} \text{ a finite sequence of } R\text{-names}\}$ of S , an $(M^-, R^+)^\delta$ -tidy partition, such that:

1. if $P \in \mathcal{S}^{\tilde{r}}$ and $P \xrightarrow{\alpha} P'$ where $\alpha \notin M^- \cup R^+$, then $P' \in \mathcal{S}^{\tilde{r}}$;
2. if $P \in \mathcal{S}^{\tilde{r}}$ and $P \xrightarrow{\mu} P'$ where $\mu \in M^-$ and $r = \text{obj}_R(\mu)$, then $P' \in \mathcal{S}^{\tilde{r}r}$;
3. if $P \in \mathcal{S}^{\tilde{r}}$ and $P \xrightarrow{\rho} P'$ where $\rho \in R^+$, then $\text{subj}(\rho) = r \in \tilde{r}$ and $P' \in \mathcal{S}^{\tilde{r}-r}$.

Further, S is $(M^-, R^+)^\delta$ -ready if it is $(M^-, R^+)^\delta$ -tidy and

- 4a. if $P \in \mathcal{S}^{\tilde{r}}$ and $\rho \in R^+$ with $\text{subj}(\rho) \in \tilde{r}$, then $P \xrightarrow{\rho}$;
- 4b. if $P \xrightarrow{\rho_1} \simeq P'_1 \xrightarrow{\rho_2} \simeq P_1$ and $P \xrightarrow{\rho_2} \simeq P'_2 \xrightarrow{\rho_1} \simeq P_2$ where $\rho_1, \rho_2 \in R^+$, then $P_1 \simeq P_2$.

Similarly, we define a $(M^+, R^-)^\delta$ -tidy (partition) and say \mathcal{S} is $(M^+, R^-)^\delta$ -disciplined if it is $(M^+, R^-)^\delta$ -tidy with $(M^+, R^-)^\delta$ -tidy partition $\{\mathcal{S}^r\}_{\bar{r}}$ and

- 4c. if $P \in \mathcal{S}^r$ (where r is a singleton) and $P \xrightarrow{\alpha}$ where $\alpha \in M^+$ then $P \Rightarrow \xrightarrow{\beta}$ for some β with $\text{subj}(\beta) = \bar{r}$.

The main result asserts that a ready agent cannot be distinguished in a disciplined context from its pruned version.

Theorem 4.4.6 Suppose \mathcal{P} is an $(M^-, R^+)^\delta$ -ready set with $(M^-, R^+)^\delta$ -tidy partition $\{\mathcal{P}^{\bar{r}}\}_{\bar{r}}$, and \mathcal{T} an $(M^+, R^-)^\delta$ -disciplined set with $(M^+, R^-)^\delta$ -tidy partition $\{\mathcal{T}^{\bar{r}}\}_{\bar{r}}$. Suppose $P \in \mathcal{P}^e$, $T \in \mathcal{T}^e$ and $(\nu\tilde{z})(P \mid T)$ is M, R -closed. Then $(\nu\tilde{z})(P \mid T) \simeq (\nu\tilde{z})(P \mid T^b)$.

PROOF: Define \mathcal{B}^0 and \mathcal{B}^1 as follows. First, $(S_1, S_2) \in \mathcal{B}^0$ if $S_1 = (\nu\tilde{z})(P_1 \mid T_1)$ and $S_2 = (\nu\tilde{z}\tilde{w})(P_2 \mid T_2^b)$ where $P_2 \in \mathcal{P}^e$, $T_2 \in \mathcal{T}^e$, $P_1 \xrightarrow{s} \simeq P_2$, $T_1 \xrightarrow{\bar{s}} \simeq T_2$, $s = \rho_1 \dots \rho_n \in R^{+*}$, $\bar{s} = \bar{\rho}_1 \dots \bar{\rho}_n \in R^{-*}$ with $\rho_i = r_i\langle\tilde{y}_i\rangle$, $\bar{\rho}_i = (\nu\tilde{w}_i)\bar{r}_i\langle\tilde{y}_i\rangle$, and $\tilde{w} = \tilde{w}_1 \dots \tilde{w}_n$, and no derivative of S_1 contains a free occurrence in subject position of a name of sort in $R \cup M$.

Secondly, $(S_1, S_2) \in \mathcal{B}^1$ if $S_1 = (\nu\tilde{z})(P_1 \mid T_1)$ and $S_2 = (\nu\tilde{z}\tilde{w})(P_2 \mid T_2^b)$ where $P_2 \in \mathcal{P}^r$, $T_2 \in \mathcal{T}^r$, $P_1 \xrightarrow{s} \simeq P_2$, $T_1 \xrightarrow{\bar{s}} \simeq T_2$, $s = \rho_1 \dots \rho_n \in R^{+*}$, $\bar{s} = \bar{\rho}_1 \dots \bar{\rho}_n \in R^{-*}$ with $\rho_i = r_i\langle\tilde{y}_i\rangle$, $\bar{\rho}_i = (\nu\tilde{w}_i)\bar{r}_i\langle\tilde{y}_i\rangle$, and $\tilde{w} = \tilde{w}_1 \dots \tilde{w}_n$, and no derivative of S_1 contains a free occurrence in subject position of a name of sort in $R \cup M$.

We show that $\mathcal{B}^0 \cup \mathcal{B}^1 \cup \simeq$ is a branching bisimulation. Suppose $(S_1, S_2) \in \mathcal{B}^0$ where $S_1 = (\nu\tilde{z})(P_1 \mid T_1)$ and $S_2 = (\nu\tilde{z}\tilde{w})(P_2 \mid T_2^b)$ are as above. Suppose $S_2 \xrightarrow{\alpha} Q'_2 = (\nu\tilde{p})(P'_2 \mid T_2^b)$. Then $S_1 \Rightarrow S'_1 = (\nu\tilde{z}\tilde{w})(P'_1 \mid T'_1)$ where $P'_1 \simeq P_2$, $T'_1 \simeq T_2$ and $(S_2, Q'_2) \in \mathcal{B}^0$. Thus it is not difficult to see that $S'_1 \Rightarrow \xrightarrow{\alpha} Q_1 = (\nu\tilde{p})(P''_1 \mid T''_1)$ where $P''_1 \simeq P'_2$ and $T''_1 \simeq T'_2$ and so $(Q_1, Q_2) \in \mathcal{B}^0 \cup \mathcal{B}^1$. Now suppose $S_1 \xrightarrow{\alpha} Q_1$. There are several cases.

1. Suppose $Q_1 \equiv (\nu\tilde{z})(P'_1 \mid T_1)$ where $P_1 \xrightarrow{\alpha} P'_1$ and α is τ , $x\langle\tilde{y}\rangle$ or $(\nu\tilde{u})\bar{x}\langle\tilde{y}\rangle$ where $\tilde{u} \cap \tilde{z} = \emptyset$. Then by Lemma 4.4.4, there are P'_2, P''_2 such that $P_2 \Rightarrow P'_2 \xrightarrow{\alpha} P''_2$ and $P'_1 \xrightarrow{s} \simeq P'_2$ with $P'_2 \simeq P_2$. So $S_2 \Rightarrow (\nu\tilde{z}\tilde{w})(P'_2 \mid T_2) \xrightarrow{\alpha} (\nu\tilde{z}\tilde{w})(P''_2 \mid T_2^b)$, and it is easy to see that $(S_1, (\nu\tilde{z}\tilde{w})(P'_2 \mid T_2^b)) \in \mathcal{B}^0$ and $(Q_1, (\nu\tilde{z}\tilde{w})(P''_2 \mid T_2^b)) \in \mathcal{B}^0$.
2. Suppose $Q_1 \equiv (\nu\tilde{z})(P_1 \mid T'_1)$ where $T_1 \xrightarrow{\alpha} T'_1$ and α is τ , $x\langle\tilde{y}\rangle$ or $(\nu u)\bar{x}\langle\tilde{y}\rangle$ where $\tilde{u} \cap \tilde{z} = \emptyset$. Then by Lemma 4.4.4, there are T'_2, T''_2 such that $T_2 \Rightarrow T'_2 \xrightarrow{\alpha} T''_2$ and $T'_1 \xrightarrow{\bar{s}} \simeq T''_2$ with $T'_2 \simeq T_2$. Since these transitions involve no action in M^+ , $T_2^b \Rightarrow T'_2^b \xrightarrow{\alpha} T''_2^b$, and $(\nu\tilde{z}\tilde{w})(P_2 \mid T_2^b) \Rightarrow (\nu\tilde{z}\tilde{w})(P_2 \mid T'_2^b) \xrightarrow{\alpha} (\nu\tilde{z}\tilde{w})(P_2 \mid T''_2^b)$. Moreover $(S_1, (\nu\tilde{z}\tilde{w})(P_2 \mid T''_2^b)) \in \mathcal{B}^0$ and $(Q_1, (\nu\tilde{z}\tilde{w})(P_2 \mid T''_2^b)) \in \mathcal{B}^0$.

3. The cases when P_1 or T_1 acts alone performing an output with free names in \tilde{z} are similar.
4. If $\alpha = \tau$ and $Q_1 = (\nu \tilde{z} \tilde{u})(P'_1 | T'_1)$ where P_1 and T_1 interact via a name of sort not in R , or via a name of sort in R with P_1 sending and T_1 receiving, then we combine elements of the arguments in the cases above. Note, in particular, that if the interaction is via an M -name then the resulting pair are related by \mathcal{B}^1 .
5. Finally, suppose $\alpha = \tau$ and $Q_1 = (\nu \tilde{z} \tilde{u})(P'_1 | T'_1)$ where $P_1 \xrightarrow{\sigma} P'_1$ and $T_1 \xrightarrow{\bar{\sigma}} T'_1$ with $\sigma = r\langle\tilde{v}\rangle$, $\bar{\sigma} = (\nu \tilde{u})\bar{r}\langle\tilde{v}\rangle$. Then it is easy to see that $r \in \text{subj}(s)$. Moreover, since $T_2 \in \mathcal{T}^e$, by Lemma 4.4.4 we have that $T'_1 \xrightarrow{\bar{s'}} \simeq T_2$ where $\bar{s'} \in \bar{s} - \bar{\sigma}$.

Now consider P_1 . Since $P_1 \xrightarrow{s} \simeq P'_1$, by Clause 4(b) in the definition of a $(M^-, R^+)^{\delta}$ -ready partition, we can see that for every permutation t of the actions of s , $P_1 \xrightarrow{t} \simeq P_2$. So suppose $s = s_1 r\langle\tilde{v}\rangle s_2 r\langle\tilde{v}\rangle s_3$, where $r \notin \text{subj}(s_1)$ and $r\langle\tilde{v}\rangle \notin s_2$. Let $w = s_1 r\langle\tilde{v}\rangle r\langle\tilde{u}\rangle s_2 s_3$. Then $P_1 \xrightarrow{w} \simeq P_2$ and by Lemma 4.4.4, $P'_1 \xrightarrow{w/\sigma} \simeq P_2$. Moreover, by the observation above and since s' is a permutation of w/σ , $P'_1 \xrightarrow{s'} \simeq P_2$. So $Q_1 \longrightarrow \simeq S_2 \equiv (\nu \tilde{z} \tilde{u})(P_2 | T_2)$ and $(Q_1, S_2) \in \mathcal{B}^0$.

Now suppose $(S_1, S_2) \in \mathcal{B}^1$ where $S_1 = (\nu \tilde{z})(P_1 | T_1)$ and $S_2 = (\nu \tilde{z} \tilde{u})(P_2 | T_2)$ are as in the definition with $P_2 \in \mathcal{P}^r$ and $T_2 \in \mathcal{T}^r$. Suppose $S_2 \xrightarrow{\alpha} Q_2$. As before, $S_1 \Rightarrow Q'_1 \xrightarrow{\alpha} Q_1$ where $(Q'_1, S_2) \in \mathcal{B}^1$ and $(Q_1, Q_2) \in \mathcal{B}^0 \cup \mathcal{B}^1$. So suppose $S_1 \xrightarrow{\alpha} Q_1$. We can carry through the same case analysis as above except for the last two cases which we consider below.

1. Suppose that $\alpha = \tau$ and $Q_1 \equiv (\nu \tilde{z} \tilde{u})(P'_1 | T'_1)$ where $P_1 \xrightarrow{\bar{\gamma}} P'_1$ and $T_1 \xrightarrow{\gamma} T'_1$ where $\gamma = m\langle\tilde{v}\rangle$ and $\bar{\gamma} = (\nu \tilde{u})\bar{m}\langle\tilde{v}\rangle$ where m is an M -name. In this case, the transitions $T_2^b \Rightarrow T_2''^b \xrightarrow{\gamma} T_2'^b$ above can not be found as $T_2 \in \mathcal{T}^r$.

Since $T_1 \xrightarrow{\bar{s}} \simeq T_2$, by Lemma 4.4.4 there are T_0, T'_2 such that $T_2 \Rightarrow T_0 \xrightarrow{\gamma} T'_2$ and $T'_1 \xrightarrow{\bar{s}} \simeq T'_2$ with $T_0 \simeq T_2$. Since \mathcal{T} is $(M^+, R^-)^{\delta}$ -disciplined, $T_0 \in \mathcal{T}^r$ and $\gamma \in M^+$, $T_0 \xRightarrow{\bar{\beta}}$ with $\text{subj}(\bar{\beta}) = \bar{\gamma}$, say $\bar{\beta} = (\nu \tilde{x})\bar{r}\langle\tilde{y}\rangle$. Hence $T_2 \xRightarrow{\bar{\beta}} T_3$ for some T_3 . Hence as T_2 is R^{δ} -confluent, $T_3 \Rightarrow T_4 \xrightarrow{\gamma} T_5$ and $T'_2 \xrightarrow{\bar{\beta}} \simeq T_5$ for some T_4 and T_5 with $T_4 \simeq T_3$. Now $T_3, T_4 \in \mathcal{T}^e$ and $T_5 \in \mathcal{T}^{r'}$ where $r' = \text{obj}_R(\gamma)$, so the transitions $T_2^b \Rightarrow T_3^b \xRightarrow{\bar{\beta}} T_4^b \xrightarrow{\gamma} T_5^b$ are possible.

Now since $P_1 \xrightarrow{s} \simeq P_2$, by Lemma 4.4.4 there are P_0 and P'_2 such that $P_2 \Rightarrow P_0 \xrightarrow{\bar{\gamma}} P'_2$, $P_0 \simeq P_2$ and $P'_1 \xrightarrow{s} \simeq P'_2$. Because \mathcal{P} is $(M^-, R^+)^{\delta}$ -ready, $P_2 \in \mathcal{P}^r$

and $\text{subj}(\beta) = r$ where $\beta = r\langle\tilde{y}\rangle$, $P_2 \xrightarrow{\beta} P_3$ for some P_3 . Since P_2 is R^δ -confluent, $P_3 \Rightarrow P_4 \xrightarrow{\tilde{\gamma}} P_5$ and $P'_2 \xrightarrow{\beta} P_5$ for some P_4 and P_5 with $P_4 \simeq P_3$. Thus $(\nu\tilde{z}\tilde{w})(P_2 \mid T_2^b) \xRightarrow{\tau} (\nu\tilde{z}\tilde{w}\tilde{x})(P_3 \mid T_3^b) \Rightarrow (\nu\tilde{z}\tilde{w}\tilde{x})(P_4 \mid T_4^b) \xrightarrow{\tau} (\nu\tilde{z}\tilde{w}\tilde{x}\tilde{u})(P_5 \mid T_5^b)$.

It remains to note that by the construction $(S_1, (\nu\tilde{z}\tilde{w}\tilde{x})(P_4 \mid T_4^b)) \in \mathcal{B}^0$ and moreover $(Q_1, (\nu\tilde{z}\tilde{u}\tilde{w}\tilde{x})(P_5 \mid T_5^b)) \in \mathcal{B}^1$. These claims follow from Lemma 4.4.4.

2. Finally, suppose $\alpha = \tau$ and $Q_1 = (\nu\tilde{z}\tilde{u})(P'_1 \mid T'_1)$ where $P_1 \xrightarrow{\sigma} P'_1$ and $T_1 \xrightarrow{\bar{\sigma}} T'_1$ with $\sigma = r\langle\tilde{v}\rangle$, $\bar{\sigma} = (\nu\tilde{u})\bar{r}\langle\tilde{v}\rangle$. Two cases exist: if $T'_1 \xrightarrow{\bar{\sigma}-\bar{\sigma}} T_2$ then the result follows as in case 5 above. Otherwise, by Lemma 4.4.4, $T_2 \Rightarrow \xrightarrow{\bar{\sigma}} T'_2$ and $P_2 \Rightarrow \xrightarrow{\sigma} P'_2$, where $T'_1 \xrightarrow{\bar{\sigma}} T'_2$ and $P'_1 \xrightarrow{\sigma} P'_2$. So $S_2 \Rightarrow Q'_2 \xrightarrow{\tau} Q_2 = (\nu\tilde{z}\tilde{u}\tilde{w})(P'_2 \mid T_2^b)$ where $(S_1, Q'_2) \in \mathcal{B}^1$. Since $T_2 \in \mathcal{T}^e$, using arguments similar to those in 5 above, we may deduce that $(Q_1, Q_2) \in \mathcal{B}^0$. \square

The extension of social confluence relaxing the condition on the uniqueness of names follows in a very similar manner. Note that it is not required to impose any further conditions on a $(Q^-, A^+)^\delta$ -client (as was the case with a (M^-, R^+) -ready partition) since commutativity of actions is ensured by the definition of a $(Q^-, A^+)^\delta$ -base.

Definition 4.4.7 Let Q and A be distinct sorts. A process P is $(Q, A)^\delta$ -socially confluent if it is Q^δ -confluent, A^δ -confluent and for every derivative R of P the following hold:

1. if $R \xrightarrow{\alpha} R_1 \xrightarrow{\beta} R_2$ where either $\alpha \notin Q^\pm \cup \{\tau\}$ or $\beta \notin A^\pm \cup \{\tau\}$, then $R \xrightarrow{\beta} R'_1 \xrightarrow{\alpha} R'_2$ where $R_2 \simeq R'_2$;

Definition 4.4.8 Suppose λ is a Q, A -sorting. A derivation-closed set \mathcal{S} of $(Q, A)^\delta$ -socially confluent processes is a $(Q^-, A^+)^\delta$ -base if there is a partition $\{\mathcal{S}^{\tilde{a}} \mid \tilde{a} \text{ a finite sequence of } A\text{-names}\}$ of \mathcal{S} , a $(Q^-, A^+)^\delta$ -base partition, which is $(Q^-, A^+)^\delta$ -tidy. A $(Q^+, A^-)^\delta$ -base is defined dually.

Further, \mathcal{S} is a $(Q^+, A^-)^\delta$ -server if it is a $(Q^+, A^-)^\delta$ -base and

2. if $R \in \mathcal{R}^a$ (where a is a singleton) then either $R \Rightarrow \xrightarrow{\alpha}$ or $R \Rightarrow \xrightarrow{\tau} R' \Rightarrow \xrightarrow{\alpha}$ for some $\alpha = \bar{a}\langle\tilde{v}\rangle$;
3. if $R \in \mathcal{R}^{\tilde{a}}$, R is A^- -inert and $R \Rightarrow \xrightarrow{\tau} R'$, then there exists $\alpha \in A^-$ such that $R' \Rightarrow \xrightarrow{\alpha} R''$ and R'' is A^- -inert;
4. if $R \in \mathcal{R}^e$, $R \xrightarrow{u} R_1$ where $\gamma \in \{\tau\} \cup A^-$ and $u \in Q^{+*}$, then there exists $\beta \in u$ such that $R \xrightarrow{\beta} R_2$ and $R_2 \xrightarrow{u/\beta} R_1$;

5. if $R \in \mathcal{R}^a$, $R \xrightarrow{\tau} \simeq R_1$ and $R \xrightarrow{\tau} \simeq R_2$, then $R_1 \simeq R_2$.

Finally, \mathcal{S} is a $(Q^-, A^+)^\delta$ -client if it is a $(Q^-, A^+)^\delta$ -base with a $(Q^-, A^+)^\delta$ -base partition $\mathcal{S}^{\tilde{a}}$ and

6. if $E \in \mathcal{E}^{\tilde{a}}$, $\alpha \in A^+$ and $\text{subj}(\alpha) \in \tilde{a}$, then $E \xrightarrow{\alpha}$.

The definitions are identical to those of a (Q^+, A^-) -server and a (Q^-, A^+) -client apart from the fact that in the new definitions indices are sequences as opposed to sets and the agents are Q^δ -confluent and A^δ -confluent. We have the following theorem:

Theorem 4.4.9 Suppose \mathcal{R} is a $(Q^+, A^-)^\delta$ -server system with a $(Q^+, A^-)^\delta$ -server partition $\{\mathcal{R}^{\tilde{a}}\}_{\tilde{a}}$ and \mathcal{E} is a $(Q^-, A^+)^\delta$ -client system with $(Q^-, A^+)^\delta$ -client partition $\{\mathcal{E}^{\tilde{a}}\}_{\tilde{a}}$. Suppose $R_1 \in \mathcal{R}^a$, $E_1 \in \mathcal{E}^a$ and $R_1 \xrightarrow{\bar{\alpha}} R_2$, $E_1 \xrightarrow{\alpha} E_2$ where $\text{subj}(\alpha) = a$, where $\alpha \text{ comp } \bar{\alpha}$. Further suppose $(\nu \tilde{z})(E_1 \mid R_1^b)$ and $(\nu \tilde{z} \tilde{u})(E_2 \mid R_2^b)$ are Q, A -closed. Then $(\nu \tilde{z})(E_1 \mid R_1^b) \simeq (\nu \tilde{z} \tilde{u})(E_2 \mid R_2^b)$, where $\tilde{u} = \text{bn}(\bar{\alpha})$.

PROOF: The proof follows using arguments similar to those of Theorem 4.3.7. \square

We conclude with the main theorem.

Theorem 4.4.10 Suppose \mathcal{R} is a $(Q^+, A^-)^\delta$ -server system with a $(Q^+, A^-)^\delta$ -server partition $\{\mathcal{R}^{\tilde{a}}\}_{\tilde{a}}$ and \mathcal{E} is a $(Q^-, A^+)^\delta$ -client system with $(Q^-, A^+)^\delta$ -client partition $\{\mathcal{E}^{\tilde{a}}\}_{\tilde{a}}$. Suppose $R \in \mathcal{R}^e$, $E \in \mathcal{E}^e$ and $(\nu \tilde{z})(E \mid R)$ is Q, A -closed. Then $(\nu \tilde{z})(E \mid R) \simeq (\nu \tilde{z})(E \mid R^b)$.

PROOF: The proof is similar to that of Theorem 4.3.8 but makes additional use of the commutativity of actions as captured in the definitions of $(Q^+, A^-)^\delta$ -social confluence. Let $(S_1, S_2) \in \mathcal{B}$ if $S_1 = (\nu \tilde{z})(E_1 \mid R_1)$ and $S_2 = (\nu \tilde{z}')(E_2 \mid R_2^b)$ where $E_2 \in \mathcal{E}^e$, $R_2 \in \mathcal{R}^e$ and there exist E, R, s, u, \tilde{a} , such that $R \in \mathcal{R}^{\tilde{a}}$, $E \in \mathcal{E}^{\tilde{a}}$, R is A^- -inert and

$$\begin{aligned} E_1 &\xrightarrow{s} \simeq E, & R_1 &\xrightarrow{\tilde{s}} \simeq R, & \tilde{s} &= \overline{\alpha_1} \dots \overline{\alpha_n}, & \overline{\alpha_i} &= (\nu \tilde{p}_i) \overline{r_i}(\tilde{v}_i), & s &\text{comp } \tilde{s} \\ E_2 &\xrightarrow{\tilde{u}} \simeq E, & R_2 &\xrightarrow{u} \simeq R, & \tilde{u} &= \overline{\beta_1} \dots \overline{\beta_k}, & \overline{\beta_i} &= (\nu \tilde{y}_i) \overline{q_i}(\tilde{x}, b_i), & u &\text{comp } \tilde{u}, \end{aligned}$$

where $r_i, b_i : A, q_i : Q, b_i \in \tilde{y}_i$, and $\tilde{a} = \{b_1, \dots, b_k\}$. We show that $\mathcal{B} \simeq \cup \simeq$ is a branching bisimilarity.

First suppose $(S_1, S_2) \in \mathcal{B}$ where $S_1 = (\nu \tilde{z})(E_1 \mid R_1)$ and $S_2 = (\nu \tilde{z}')(E_2 \mid R_2^b)$ are as above. Suppose $S_2 \xrightarrow{p} Q_2$. Various possibilities exist and using the same arguments as in the proof of Theorem 4.3.8 we may find that $S_1 \xRightarrow{p} Q_1$ where $(Q_1, Q_2) \in \mathcal{B} \simeq$.

Now suppose $(S_1, S_2) \in \mathcal{B}$ and $S_1 \xrightarrow{p} Q_1$. There are several cases and they may all be handled as in the proof of Theorem 4.3.8 with one exception:

Suppose $\alpha = \tau$ and $Q_1 = (\nu \tilde{z} \tilde{x})(E'_1 \mid R'_1)$ where $E_1 \xrightarrow{\gamma} E'_1$, $R_1 \xrightarrow{\bar{\gamma}} R'_1$ and $\gamma = a(\tilde{y})$, $\bar{\gamma} = (\nu \tilde{x})\bar{a}(\tilde{y})$, $a : A$. Then $\gamma \in s$ as otherwise, by A^δ -confluence and Lemma 4.4.4, $R \xrightarrow{\bar{\gamma}}_{\simeq} R'$ which contradicts the construction of \mathcal{B} and in particular that of R . So suppose $\gamma = \alpha_i$ where $\alpha_j \neq \gamma$ for all $j < i$ (recall that $s = \alpha_1 \dots \alpha_n$). Then by Definition 4.4.7(1), $E_1 \xrightarrow{\gamma}_{\simeq} \xrightarrow{s/\gamma}_{\simeq} E$ and so $E'_1 \xrightarrow{s/\gamma}_{\simeq} E$. Moreover, by A^δ -confluence and Lemma 4.4.4, $R'_1 \xrightarrow{s'}_{\simeq} R$ where $s' \in \bar{s} - \bar{\gamma}$. Then by Definition 4.4.7(1), $R'_1 \xrightarrow{s'/\bar{\gamma}}_{\simeq} R$. So since $E'_1 \xrightarrow{s/\gamma}_{\simeq} E$ and $R'_1 \xrightarrow{s'/\bar{\gamma}}_{\simeq} R$ $(Q_1, S_2) \in \mathcal{B}$ as required.

Thus the proof may be completed using a similar argument to that in Theorem 4.3.8. \square

Chapter 5

Concurrent Operations on *B*-trees

Concurrency control is the activity of preventing interference among concurrent processes, or transactions, accessing a data structure. As multiprocessor configurations have become widespread, concurrency control has been a subject of much research. One of the main challenges has been to design transaction-processing algorithms in which the control is sufficient to guarantee correctness while allowing as much concurrency as possible. Besides correctness and efficiency, such algorithms need to provide high availability when parts of the system fail and good recovery mechanisms. A great variety of algorithms have been proposed and implemented. These algorithms involve several different techniques and many are associated with particular data structures or are tailored to particular architectures and applications. Considering the complexity of the issues involved, it is not surprising that many of the algorithms developed have complicated and subtle behaviours and that it is difficult to give convincing informal arguments of their correctness. In fact, without a formal model of the system in question it is often difficult to even formulate precisely what the correctness properties are. Thus the need for formal modelling of transaction-processing algorithms was very early recognized.

The classical approaches to database concurrency control use a syntactic criterion of correctness known as *conflict-preserving serializability*. A concurrent execution of a system is conflict-preserving serializable if there exists a serial execution of the same system which performs the same operations on the database and orders conflicting operations in the same way. While this criterion is suitable for algorithms in some settings, it appears to be unduly restrictive and even inappropriate in many others. As a result, the theory has been extended and modified

to encompass the different settings and has been used to produce several different definitions of correctness for different types of algorithms. For an overview of a large body of work concerning serializability see [BHG87, Pap86]. With the intention of stating a general correctness criterion independent of algorithm and system details, a theory of semantic serializability has also been explored and developed. Kung and Papadimitriou, [KP83], initiated the theory of a semantic approach to concurrency control by studying the relationship between the information available to a concurrency control algorithm and the achievable concurrency and subsequently, various other authors have investigated the possibility of employing semantic information for enunciating criteria of correctness [Cas81, GM84, ST85, Lam86, SG88].

In this chapter we perform a rigorous analysis of concurrent operations on a dynamic data structure, a variant of the B -tree. The B -tree [BM72] and variants of it are widely used as index structures. Concurrent operations on such structures have been studied in many papers including [BS77, KW82, LY81, MS78, Sag86, Sam76, SG88]. In particular, we study a variant of the B^* -tree [Wed74]. This variant is the B^{link} -tree of [LY81], and the operations we consider are those of [Sag86]. These operations improve on those of [LY81] in that an insertion process locks at most one tree-node at any time, as opposed to two or three, and in that compression of the structure is effected to redistribute data following deletions.

By using the π_v -calculus as a basis of our analysis, we may give direct and succinct descriptions of the operations and the underlying data structure which capture naturally its dynamically-evolving interconnection. In addition, we are able to enunciate a correctness criterion and perform rigorous proofs of correctness of the operations which provide greater confidence than is afforded by the kinds of techniques used in the papers cited above. Further, we believe that the proofs here are perspicuous and give valuable insight into why the operations are correct. In contrast with forms of serializability, the correctness of the system is expressed in terms of its observable behaviour. The intended behaviour of the system is captured by a simple agent and the criterion of correctness is that the two systems are behaviourally equivalent. The principle that concurrent systems should be compared in terms of their observable behaviour is indeed widely held and has been elaborated in concurrency theory. The approach has also been explored and shown to be sound in the setting of database concurrency control in [LMWF94], where I/O-automata are employed to give a uniform analysis of a variety of concurrency control algorithms. Additional advantages of the use of a mobile calculus for the analysis are that it enables a direct and natural representation of the evolving interconnection of the data structure and it allows application of the rich process-calculus techniques

to reason about the system.

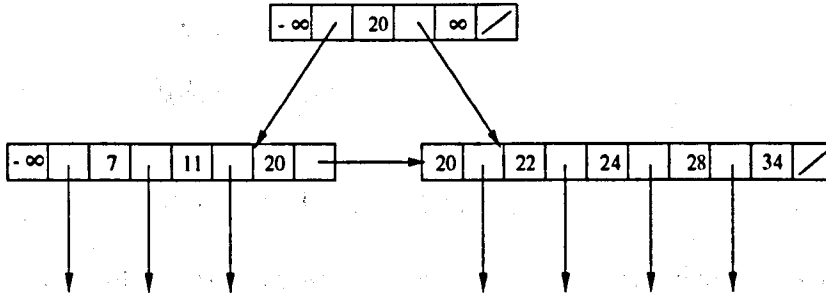
The theory of partial confluence plays a central rôle in the correctness proof. First, the operations of [LY81, Sag86] are such that a searching process may access a tree-node even when it is locked by an inserting process. Using the theory of partial confluence, we show that in order to reason about such operations it suffices to analyse a simpler system in which at most one process may access a node at any one time. Secondly, the theory of social confluence enables us to show that in a certain class of contexts, the behaviour of the structure is indistinguishable from a simplification of it in which at most one operation is active at any one time. We are thus able to restrict attention to that simpler system.

The following section contains a description of the sequential index structure and the operations on it. Section 5.2 presents the correctness proof while Section 5.3 generalizes the results to the case where the underlying structure allows multiple searches and a single insertion to take place in a node concurrently. We then discuss the deletion and compression operations of [Sag86]. Utilizing insights gained from the previous analysis, we propose an improvement to the latter in Section 5.4. We also discuss how a correctness proof may be obtained by extending the existing analysis.

5.1 The structure

This section contains the process-calculus descriptions of the B^{link} -tree [LY81] and the concurrent operations on it presented in [Sag86]. We begin with a brief, informal description of the structure. A B^{link} -tree indexes a database by storing in its leaves pairs $\langle k, b \rangle$ with k an integer key associated with a record and b a pointer to the database record. All of its leaves are at the same distance from the root. Each of its nodes has j values (we use ‘value’ and ‘key’ interchangeably) and j pointers where $2 \leq j \leq 2m + 1$ if the node is the root and $m + 1 \leq j \leq 2m + 1$ otherwise (for some tree-parameter m). A node’s integer values are stored in ascending order. It is intended that a node’s greatest value, its *high key*, is the largest key in the subtree rooted at the node. All but the last pointer of a node point to children of the node. All but the last of a leaf’s pointers point to records of the database. The last pointer of a node or leaf, its *link*, points to the next node at the same level of the tree (if it exists). The purpose of links is to provide additional paths through the structure. The rightmost node at each level has high key ∞ and link nil. If a node has keys $\tilde{k} = k_1, \dots, k_j$ and pointers $\tilde{p} = p_1, \dots, p_j$, an intended invariant is that for $i < j$, pointer p_i points to the subtree whose leaves contain all keys k with $k_i < k \leq k_{i+1}$.

An example of a B^{link} -tree is given in the following figure.



We now proceed to the process-calculus description. The process-calculus sorting is important in organizing it. The sorts utilized by the definition are the following, where I , S stand for 'insert' and 'search'.

1. The sorts are

| | |
|-------|--|
| P^m | where $m \in \{I, S\}$, invocation links, |
| R^m | where $m \in \{I, S\}$, return links, |
| Q_m | $m \in \{I, S\}$, question links, |
| A_m | $m \in \{I, S\}$, answer links, |
| B | database links, |
| S | store links, and |
| L | level links. |

- The basic types are **bool**, **int** and **signal**. The values of type **signal** include *link* and *done*.
- The record type $\{ins \vdash P^I, srch \vdash P^S\}$ is abbreviated to P_N . Terms of this type will be process calculus representations of pointers to nodes and we will refer to them as *node names*. If p is a node name, $p * ins$ and $p * srch$ select its two name-components via which insertions and searches may be initiated.

The sorting λ is the following partial function on link sorts where B is the sort of the names representing pointers to database records.

$$\begin{aligned}
 \lambda(P^S) &= \{(int, R^S)\} \\
 \lambda(P^I) &= \{(int, P_N, R^I), (int, B, R^I)\} \\
 \lambda(R^S) &= \{(P_N), (signal, P_N), (signal, B)\} \\
 \lambda(R^I) &= \{(), (P_N), (P_N, int)\}
 \end{aligned}$$

$$\begin{aligned}
\lambda(Q_S) &= \{(int, A_S)\} \\
\lambda(Q_I) &= \{(int, B, A_I)\} \\
\lambda(A_S) &= \{(B)\} \\
\lambda(A_I) &= \{()\} \\
\lambda(S) &= \{(P_N), (P_N, L)\} \\
\lambda(L) &= \{(P_N^k) | k \geq 1\}.
\end{aligned}$$

The significance of the sorting will become clear when the definitions are given. Briefly, it is based on the following ideas. The sorting decrees that a name of sort P^S carries a pair consisting of an integer (a key to be searched for) and a name of sort R^S (via which the result of the search should be returned). Similarly, a name of sort P^I carries a triple consisting of a key, a pointer to a tree node or to a database record (to be inserted) and a name of sort R^I (via which the completion of the insertion should be communicated). On the other hand, a name of sort R^S may carry either a pointer or a pair consisting of a signal and a pointer. The other sorts are understood similarly.

We make use of a function $find_N$ which given a triple $(k, \tilde{p}, \tilde{k})$ as argument, where \tilde{k} is an ordered sequence, returns the pointer p_1 if $k \leq k_2$, the pointer p_i if $k_i < k \leq k_{i+1}$ and $i > 2$, and the link last \tilde{p} otherwise:

$$find_N(k, \tilde{p}, \tilde{k}) = \begin{cases} p_1, & \text{if } k \leq k_2 \\ p_i, & \text{if } k_i < k \leq k_{i+1}, i > 2 \\ \text{last } \tilde{p}, & \text{otherwise.} \end{cases}$$

The definition of the agent $NODE \equiv NODE(p, \tilde{k}, \tilde{p})$ representing a non-leaf, non-root node with name p of sort P_N , storing keys \tilde{k} and pointers \tilde{p} follows below. For clarity, we write *notfull* for $\# \tilde{k} < 2m + 1$ and *full* for $\# \tilde{k} = 2m + 1$.

$$\begin{aligned}
NODE \stackrel{\text{def}}{=} & p * srch(k, r). \text{ cond } (k > \text{last } \tilde{k} \triangleright \bar{r} \langle link, find_N(k, \tilde{p}, \tilde{k}) \rangle. NODE, \\
& k \leq \text{last } \tilde{k} \triangleright \bar{r} \langle find_N(k, \tilde{p}, \tilde{k}) \rangle. NODE) \\
& + p * ins(k, q, r). \text{ cond } (k > \text{last } \tilde{k} \triangleright \bar{r} \langle find_N(k, \tilde{p}, \tilde{k}) \rangle. NODE, \\
& k \in \tilde{k} \triangleright \bar{r}. NODE(p, \tilde{k}, \tilde{p}''), \\
& \text{notfull} \triangleright \bar{r}. NODE(p, \tilde{k}', \tilde{p}'), \\
& \text{full} \triangleright (\nu p')(\bar{r} \langle p', k_{m+1} \rangle. \\
& (NODE(p, \tilde{k}_1, \tilde{p}_1) \mid NODE(p', \tilde{k}_2, \tilde{p}_2)))
\end{aligned}$$

where

$$\tilde{p}'' = p_1 \dots p_{h-1} q p_{h+1} \dots p_j, \text{ if } k = k_h$$

$$\begin{aligned}
\tilde{k}', \tilde{p}' &= k_1 \dots k_h k k_{h+1} \dots k_j, p_1 \dots p_h q p_{h+1} \dots p_j, \text{ if } k_h < k < k_{h+1} \\
\tilde{k}_1, \tilde{p}_1 &= \begin{cases} k_1 \dots k_{m+1}, p_1 \dots p_m p', & \text{if } k > k_{m+1} \\ \text{last } k_1 \dots k_q k k_{q+1} \dots k_{m+1}, p_1 \dots p_q p p_{q+1} \dots p_m p', & \text{if } k_q < k < k_{q+1} \leq k_{m+1} \end{cases} \\
\tilde{k}_2, \tilde{p}_2 &= \begin{cases} k_{m+1} \dots k_{2m+1}, p_{m+1} \dots p_{2m+1}, & \text{if } k < k_{m+1} \\ k_{m+1} \dots k_q k k_{q+1} \dots k_{2m+1}, p_{m+1} \dots p_q p p_{q+1} \dots p_{2m+1}, & \text{if } k_{m+1} \leq k_q < k < k_{q+1}. \end{cases}
\end{aligned}$$

Thus in its quiescent state a node may accept a search request via the name $p * srch$ and an insertion request via the name $p * ins$. In response to a search request for a key k it returns the pointer p_j such that $k_j < k \leq k_{j+1}$ (as specified by $find_N$). If this pointer is the link of the node, an indication of this fact is sent as well. Suppose insertion of k, q is requested with return name r thus: $NODE \xrightarrow{p * ins(k, q, r)} P$ where P is the continuation agent. If k is greater than the high key of the node then the node returns its link and resumes its quiescent state. Otherwise, if k is one of the node's keys then the pointers of the node are updated accordingly. If $k \notin \tilde{k}$ and the node is not full, the pair is inserted and a signal is sent via r indicating completion of the insertion. Otherwise, the node is split, represented by the concurrent composition of two nodes: $NODE(p, \tilde{k}_1, \tilde{p}_1)$ and $NODE(p', \tilde{k}_2, \tilde{p}_2)$ where p' is a new name. The data of the original node is divided between the two. The high key of the first node becomes the value k_{m+1} which is also the first key of the second node, and its link becomes name p' . The second node which contains the larger keys of the original node also assumes its link and high key. Via r is returned the pair $\langle p', k_{m+1} \rangle$ consisting of the name of the new node and its smallest key k_{m+1} . The recipient of this pair, the agent responsible for initiating the insertion, will use it to add a new pointer to the tree: see below.

The definition of the agent LEAF is similar. It is given by the abstraction $LEAF \equiv LEAF(p, \tilde{k}, \tilde{b}, q)$ where $p : P_N$ is the name of the leaf, \tilde{k} are the integer values stored in the leaf, \tilde{b} are pointers to records of the database and q is the link pointer of the leaf. It uses the following function:

$$find_L(k, \tilde{b}, \tilde{k}) = \begin{cases} b_i, & \text{if } k = k_{i+1} \\ \text{nil}, & \text{otherwise} \end{cases}$$

The definition of LEAF is as follows:

$$\begin{aligned}
\text{LEAF} \stackrel{\text{def}}{=} & p * \text{srch}(k, r). \text{cond} (k > \text{last } \tilde{k} \triangleright \bar{r} \langle \text{link}, q \rangle. \text{LEAF}, \\
& k \leq \text{last } \tilde{k} \triangleright \bar{r} \langle \text{done}, \text{find}_L(k, \tilde{k}, \tilde{b}) \rangle. \text{LEAF}) \\
& + p * \text{ins}(k, b, r). \text{cond} (k > \text{last } \tilde{k} \triangleright \bar{r} \langle q \rangle. \text{LEAF}, \\
& k \in \tilde{k} \triangleright \bar{r}. \text{LEAF} \langle p, \tilde{k}, \tilde{b}''q \rangle, \\
& \text{notfull} \triangleright \bar{r}. \text{LEAF} \langle p, \tilde{k}', \tilde{b}' \rangle, \\
& \text{full} \triangleright (\nu p') (\bar{r} \langle p', k_{m+1} \rangle. (\text{LEAF} \langle p, \tilde{k}_1, \tilde{b}_1 p' \rangle \\
& \quad | \text{LEAF} \langle p', \tilde{k}_2, \tilde{b}_2 q \rangle))
\end{aligned}$$

where

$$\begin{aligned}
\tilde{b}'' &= b_1 \dots b_{h-1} q b_{h+1} \dots b_j, \text{ if } k = k_h \\
\tilde{k}', \tilde{b}' &= k_1 \dots k_h k k_{h+1} \dots k_j, \quad b_1 \dots b_{h-1} b b_h \dots b_j, \text{ if } k_h < k < k_{h+1} \\
\tilde{k}_1, \tilde{b}_1 &= \begin{cases} k_1 \dots k_{m+1}, \quad b_1 \dots b_m b', \text{ if } k > k_{m+1} \\ k_1 \dots k_q k k_{q+1} \dots k_{m+1}, \quad b_1 \dots b_{q-1} b b_q \dots b_m b', \\ \quad \text{if } k_q < k < k_{q+1} \leq k_{m+1} \end{cases} \\
\tilde{k}_2, \tilde{b}_2 &= \begin{cases} k_{m+1} \dots k_{2m+1}, \quad b_{m+1} \dots b_{2m+1} \text{ if } k < k_{m+1} \\ k_{m+1} \dots k_q k k_{q+1} \dots k_{2m+1}, \quad b_{m+1} \dots b_{q-1} b b_q \dots b_{2m+1}, \\ \quad \text{if } k_{m+1} \leq k_q < k < k_{q+1}. \end{cases}
\end{aligned}$$

Agent LEAF behaves similarly to agent NODE. The main difference between their behaviours is that when a search terminates successfully at a leaf, the signal *done* is returned via name *r* along with the result, a name of sort B, to indicate completion. Moreover, a NODE and a LEAF have differences in structure. The keys of agent $\text{LEAF} \langle p, \tilde{k}, \tilde{b}q \rangle$ signify the following: k_1 is a key associated with the leaf and represents its smallest key (we may see from the definition that k_1 coincides with the largest key of the leaf's left neighbour), and $k_i, i > 1$, is an index of the database with b_{i-1} being the pointer to the database record corresponding to k_i .

The definition of the agent $\text{ROOT} \equiv \text{ROOT} \langle p, \tilde{k}, \tilde{p}, \text{put} \rangle$ representing a root node with name *p* of sort P_N , keys \tilde{k} , pointers \tilde{p} of sort P_N and name *put* of sort S to be used when the root is split is as follows:

$$\begin{aligned}
\text{ROOT} \stackrel{\text{def}}{=} & p * \text{srch}(k, r). \bar{r} \langle \text{find}_N(k, \tilde{p}, \tilde{k}) \rangle. \text{ROOT} \\
& + p * \text{ins}(k, p, r). \text{cond} (k \in \tilde{k} \triangleright \bar{r}. \text{ROOT} \langle p, \tilde{k}, \tilde{p}'' \rangle, \\
& \quad \text{notfull} \triangleright \bar{r}. \text{ROOT} \langle p, \tilde{k}', \tilde{p}' \rangle, \\
& \quad \text{full} \triangleright (\nu p_0, p') (\overline{put} \langle p_0 \rangle. \bar{r}. (\text{NODE} \langle p, \tilde{k}_1, \tilde{p}_1 \rangle \\
& \quad \quad | \text{NODE} \langle p', \tilde{k}_2, \tilde{p}_2 \rangle | \text{NEWROOT}))
\end{aligned}$$

where

$$\text{NEWROOT} \equiv \text{ROOT} \langle p_0, \langle -\infty, k_{m+1}, \infty \rangle, \langle p, p', \text{nil} \rangle, \text{put} \rangle$$

and $\tilde{k}', \tilde{p}', \tilde{p}'', \tilde{k}_1, \tilde{k}_2, \tilde{p}_1$ and \tilde{p}_2 are as for NODE. This third clause of the conditional describes the behaviour of a full root when an insertion is requested. The root assumes node status and a new sibling node (with new name p') and a new root (with new name p_0) are created, the latter with pointers to the old root and its new sibling. Note that the restriction $(\nu p_0, p')$ ensures that the names are indeed new and different from all others. Before the three nodes become accessible, the name of the new root is sent via name *put* to the following agent, $\text{STORE} \equiv \text{STORE}(\tilde{p}, \text{get}, \text{put}, \text{que})$ which is responsible for recording the names of the current and previous roots:

$$\text{STORE} \stackrel{\text{def}}{=} \overline{\text{get}}(\text{last } \tilde{p}).\text{STORE} + \text{put}(p_0).\text{STORE}(\tilde{p}p_0, \dots) + \text{que}(p, l).\bar{l}(\tilde{p}_1).\text{STORE}$$

where in the last summand $\tilde{p} = \tilde{p}_2 p \tilde{p}_1$. Here names *get*, *put* and *que* are of sort S and name *l* is of sort L. Further, \tilde{p} is the list of names (in order of creation) which have been roots of the tree, last \tilde{p} thus being the name of the current root. The names *get* and *put* are links via which the name of the current root can be read and updated. Finally, *que* may be used to discover which links named the root after the parameter p did. The B^{link} -tree in its initial state is represented by the agent

$$T_0 \stackrel{\text{def}}{=} (\nu p, p', \text{put})(\text{STORE}(\langle p \rangle, \text{get}, \text{put}, \text{que}) \\ | \text{ROOT}(p, \langle -\infty, \infty \rangle, \langle p', \text{nil} \rangle, \text{put}) | \text{LEAF}(p', \langle -\infty \rangle, \langle \text{nil} \rangle)).$$

The tree-node agents we have described are sequential in nature. They represent a structure in which at most one operation may take place in a node at any time. As mentioned in the introduction, by analysing this structure and then applying the theory of partial confluence we are able to deduce results about structures whose nodes may be accessed concurrently. This point will be discussed in Section 5.3.

It remains to describe the search and insert operations. The searcher $S \equiv S(s, \text{get})$ is defined as follows:

$$S \stackrel{\text{def}}{=} !s(k, a).\text{get}(p).\text{Search}(k, p, a) \\ \text{Search} \stackrel{\text{def}}{=} (k p a)(\nu r)\overline{p * \text{srch}}(k, r).(r(p')).\text{Search}(k, p', a) \\ + r(\text{link}, p').\text{Search}(k, p', a) \\ + r(\text{done}, b).\bar{a}(b).0.$$

The agent S may repeatedly spin off searchers when supplied via s with a key k to search for and a link a via which to return the result of the search. On initiation of a search, the searcher reads from the STORE via *get* the name of the root of the structure. It then traces a path down the tree until it reaches a LEAF which synchronizes with it by performing an action $\bar{r}(\text{done}, b)$ returning the result b of the search which is then emitted via the name a . The searcher then becomes the inactive agent 0. The inserter $I \equiv I(i, \text{get}, \text{que})$ is defined as follows:

$$\begin{aligned}
I &\stackrel{\text{def}}{=} ! i(k, b, a). \text{get}(q). \text{Down}\langle k, b, a, q, \varepsilon \rangle \\
\text{Down} &\stackrel{\text{def}}{=} (k \ b \ a \ q \ \tilde{p}) (\nu r) \overline{q * \text{srch}}\langle k, r \rangle. (r(\text{link}, q'). \text{Down}\langle k, b, a, q', \tilde{p} \rangle \\
&\quad + r(q'). \text{Down}\langle k, b, a, q', \tilde{p} \ q \rangle \\
&\quad + r(\text{done}, q'). \text{Add}\langle k, b, a, q, \tilde{p} \rangle) \\
\text{Add} &\stackrel{\text{def}}{=} (k \ b \ a \ q \ \tilde{p}) (\nu r) \overline{q * \text{ins}}\langle k, b, r \rangle. (r(p', k'). \bar{a}, \text{Up}\langle k', p', \text{hd } \tilde{p}, \text{tl } \tilde{p} \rangle \\
&\quad + r. \bar{a}. 0 \\
&\quad + r(q'). \text{Add}\langle k, b, a, q', \tilde{p} \rangle) \\
\text{Up} &\stackrel{\text{def}}{=} (k \ p \ q \ \tilde{p}) (\nu r) \overline{q * \text{ins}}\langle k, p, r \rangle. \\
&\quad \text{cond } (\tilde{p} \neq \varepsilon \triangleright (r(p', k'). \text{Up}\langle k', p', \text{hd } \tilde{p}, \text{tl } \tilde{p} \rangle, \\
&\quad \quad + r. 0, \\
&\quad \quad + r(q'). \text{Up}\langle k, p, q', \tilde{p} \rangle) \\
&\quad \tilde{p} = \varepsilon \triangleright (r(p', k'). (\nu r) \overline{q u \varepsilon}\langle q, r \rangle. r(p'). \\
&\quad \quad \text{Up}\langle k', p', \text{hd } \tilde{p}', \text{tl } \tilde{p}' \rangle, \\
&\quad \quad + r. 0, \\
&\quad \quad + r(q'). \text{Up}\langle k, p, q', \tilde{p} \rangle)).
\end{aligned}$$

The replicator I may repeatedly spin off inserters when supplied via i with a pair $\langle k, b \rangle$ to insert and a name a via which a confirmation that the insertion has been made is to be sent. The inserter obtains the name of the root from the STORE and searches until the appropriate leaf is reached. Note that the names of the rightmost nodes in the path followed are recorded in the last parameter \tilde{p} . An insertion within a leaf may result in the splitting of it. The inserter is informed of this by receiving a pair $\langle k', p' \rangle$. In such a case the continuation agent Up is responsible for inserting this pair in the level above. This process may be repeated by Up in several levels of the tree and may result in the creation of a new root. This is the reason that the node-names \tilde{p} of the rightmost nodes visited are recorded during the searching phase. It is possible that \tilde{p} may become empty although an insertion is required at a higher level of the tree: new levels may have been created after the individual inserter began its task. If this happens, the inserter queries the STORE via que to obtain the names of the leftmost nodes at each of the new levels.

The system consisting of the structure and the operations is represented by the agent

$$P_0 \stackrel{\text{def}}{=} (\nu \text{get}, \text{que})(S \mid I \mid T_0).$$

A state reachable from this has the form

$$(\nu \tilde{x})(Q_1 \mid \dots \mid Q_n \mid S \mid I \mid T)$$

where each Q_i is an individual searcher or inserter in some state and T is reachable from T_0 . It is easy to see that P_0 is a friendly agent.

5.1.1 The low key

Let A be the node $\text{NODE}\langle p, \tilde{k}, \tilde{p} \rangle$ and consider key k_1 . It can be seen by the description of the system and in particular the procedure for splitting a node that k_1 is the low key of the node; it is a lower bound of keys expected to be found in the subtree rooted at A and only operations concerning keys greater than k_1 will be directed to this part of the tree. It can also be seen that k_1 is in fact superfluous in the definitions. The reason we include it is that it facilitates the proof of correctness by allowing us to distinguish the keys accessible from a node simply by looking at its definition, rather than having to look at its neighbouring nodes in the tree. (Note that according to the definitions, the low key of a node coincides with the high key of its left neighbour.)

It is easy to see that alternative node agents may be constructed which abstract away from information concerning the low key without deviating from the initial node's observable behaviour. We consider such a definition for an internal node. It makes use of the following function:

$$\text{find}'_N(k, \tilde{p}, \tilde{k}) = \begin{cases} p_1, & \text{if } k \leq k_1 \\ p_i, & \text{if } k_i < k \leq k_{i+1} \\ \text{last } \tilde{p}, & \text{otherwise} \end{cases}$$

The definition of the agent $N \equiv N\langle p, \tilde{k}, \tilde{p} \rangle$ representing a node with name p of sort P_N , storing keys \tilde{k} and pointers \tilde{p} follows below.

$$\begin{aligned} N \stackrel{\text{def}}{=} & p * \text{srch}(k, r). \text{cond} (k > \text{last } \tilde{k} \triangleright \bar{r}(\text{link}, \text{find}'_N(k, \tilde{p}, \tilde{k})). N, \\ & k \leq \text{last } \tilde{k} \triangleright \bar{r}(\text{find}'_N(k, \tilde{p}, \tilde{k})). N) \\ & + p * \text{ins}(k, q, r). \text{cond} (k > \text{last } \tilde{k} \triangleright \bar{r}(\text{find}'_N(k, \tilde{p}, \tilde{k})). N, \\ & k \in \tilde{k} \triangleright \bar{r}. N\langle p, \tilde{k}, \tilde{p}'' \rangle, \\ & \text{notfull} \triangleright \bar{r}. N\langle p, \tilde{k}', \tilde{p}' \rangle, \\ & \text{full} \triangleright (\nu p')(\bar{r}(p', k_{m+1}). \\ & \quad (N\langle p, \tilde{k}_1, \tilde{p}_1 \rangle \mid N\langle p', \tilde{k}_2, \tilde{p}_2 \rangle)) \end{aligned}$$

where

$$\begin{aligned} \tilde{p}'' &= p_1 \dots p_h q p_{h+2} \dots p_j, \text{ if } k = k_h \\ \tilde{k}', \tilde{p}' &= k_1 \dots k_h k k_{h+1} \dots k_j, p_1 \dots p_{h+1} q p_{h+2} \dots p_j, \text{ if } k_h < k < k_{h+1} \\ \tilde{k}_1, \tilde{p}_1 &= \begin{cases} k_1 \dots k_m, p_1 \dots p_m p', & \text{if } k > k_{m+1} \\ k_1 \dots k_q k k_{q+1} \dots k_m, p_1 \dots p_{q+1} p p_{q+2} \dots p_m p', & \\ & \text{if } k_q < k < k_{q+1} \leq k_{m+1} \end{cases} \end{aligned}$$

$$\widetilde{k}_2, \widetilde{p}_2 = \begin{cases} k_{m+1} \dots k_{2m}, p_{m+1} \dots p_{2m+1} & \text{if } k < k_{m+1} \\ k_{m+1} \dots k_q k_{q+1} \dots k_{2m}, p_{m+1} \dots p_{q+1} p_{q+2} \dots p_{2m+1}, & \\ & \text{if } k_{m+1} \leq k_q < k < k_{q+1}. \end{cases}$$

It is straightforward to prove that $N\langle p, \widetilde{k}, \widetilde{p} \rangle \simeq \text{NODE}\langle p, k, \widetilde{k}, \widetilde{p} \rangle$ for any integer k . Similarly, we may define a variant leaf, $L\langle p, \widetilde{k}, \widetilde{b}q \rangle$ and a variant root, $R\langle p, \widetilde{k}, \widetilde{p} \rangle$. Letting T'_0 be the initial tree composed of these new types of nodes,

$$T'_0 \stackrel{\text{def}}{=} (\nu p, p', put) \\ (\text{STORE}\langle \langle p \rangle, get, put, que \rangle \mid R\langle p, \langle \infty \rangle, \langle p', nil \rangle, put \rangle \mid L\langle p', \langle \rangle, \langle nil \rangle \rangle).$$

Then it follows that $T_0 \simeq T'_0$. This implies that

$$P_0 \simeq (\nu get, que)(S \mid I \mid T'_0).$$

Therefore the results we will present in the following sections may be used to deduce the correctness of the operations where the underlying data structure is a tree composed of nodes that do not hold information concerning their low key.

5.2 Correctness Proof

In this section we prove the correctness of the operations. We define an agent which gives a succinct description of the intended observable behaviour of a B^{link} -tree equipped with the concurrent insertion and search operations as presented in the previous section. This agent, F , is parametrized on: the names i and s via which the operations may be initiated; a function f recording the key-pointer associations held in the leaves of the tree; a sequence ι (the *insertions*) of triples consisting of a key k , a pointer b and a name a via which a signal is to be made to the environment when the insertion of the $\langle k, b \rangle$ -pair has been completed; a sequence ι^c (the *completed insertions*) of names a whose key-pointer pairs have been inserted but which have not been used to signal this; a sequence σ (the *searches*) of pairs consisting of a key k to be searched for and a name a to be used to return to the environment the pointer found; a sequence σ^c (the *completed searches*) of pairs consisting of a name a and a pointer b found but not yet returned.

We define $F \equiv F\langle i, s, f, \iota, \iota^c, \sigma, \sigma^c \rangle$ as follows, highlighting the changes by eliding the unchanged parameters:

$$\begin{aligned}
F \stackrel{\text{def}}{=} & i(k, b, a). F\langle \dots, \iota \langle k, b, a \rangle, \dots \rangle \\
& + s(k, a). F\langle \dots, \sigma \langle k, a \rangle, \dots \rangle \\
& + \Sigma_{\langle k, b, a \rangle \in \iota} \tau. F\langle \dots, f[b/k], \iota - \langle k, b, a \rangle, \iota^c a, \dots \rangle \\
& + \Sigma_{\langle k, a \rangle \in \sigma} \tau. F\langle \dots, \sigma - \langle k, a \rangle, \sigma^c \langle a, f(k) \rangle \rangle \\
& + \Sigma_{\langle a \rangle \in \iota^c} \bar{a}. F\langle \dots, \iota^c - a, \dots \rangle \\
& + \Sigma_{\langle a, b \rangle \in \sigma^c} \bar{a}(b). F\langle \dots, \sigma^c - \langle a, b \rangle \rangle.
\end{aligned}$$

The first and second summands represent initiation of new operations, the third and fourth invisible completion of outstanding operations (with appropriate update of the association in the case of insertion), and the fifth and sixth signals of outcomes to the environment.

Let $F_0 \equiv F\langle i, s, \epsilon, \epsilon, \epsilon, \epsilon, \epsilon \rangle$ and recall the agent P_0 describing the B^{link} -tree equipped with the concurrent searchers and inserters. The result asserting the correctness of the operations is the following.

Theorem 5.2.1 $P_0 \simeq F_0$.

The theorem above is strong in the sense that it asserts that P_0 and F_0 cannot be distinguished even if different requests for operations supply the same name for return of the answer, thus allowing the possibility, for instance, that each of two processes, accessing the system, may receive the result of the other's request. It may be argued that a weaker result would be adequate:

Theorem 5.2.2 $P_0 \simeq_A F_0$.

Here \simeq_A is defined as \simeq except that it only considers input actions where the A -component is a new name. This restriction captures the assumption that when an operation is invoked in the system, the name which is supplied for return of the answer is distinct from answer names supplied in other operation invocations. It corresponds to the use of integers to distinguish 'operation instances', in [LMWF94].

Central to the proofs are the theories of partial confluence of processes: the theory of Section 4.3 is sufficient the proof of Theorem 5.2.2, whereas the theory of Section 4.4 is necessary for the proof of Theorem 5.2.1. We present the proof of the \simeq_A result and discuss how this may be extended to obtain the full result. The proof is divided into three phases which are presented in the following three sections.

5.2.1 Analysis of T_0

We begin by considering T_0 . Let \mathcal{T} be the transition system generated by T_0 and let T_0^\dagger be the state representing the truncation of a tree where at most one operation

may be active at any time. Let

$$Q_0 \stackrel{\text{def}}{=} (\nu \text{get}, \text{que})(S \mid I \mid T_0^b).$$

The following theorem states that this system is indistinguishable from P_0 .

Theorem 5.2.3 $P_0 \simeq Q_0$

PROOF: We first suggest an intuition which makes this plausible. Suppose a number of operations are activated within a tree. Since each tree node allows at most one operation at any time, it must be that the operations have been requested of distinct tree nodes. It is hence clear that the order in which they take place is immaterial, in the sense that for all possible interleavings of the operations the nodes concerned will return the same results to the operations.

The proof of the theorem is based on the partial confluence of the components and it is an easy application of the partial confluence theory. Let P be the invocation sorts P^S and P^I , of the names used by the agents S and I for reading and writing tree nodes and R be the return sorts R^S and R^I of the names used by the tree to return the results to the invocation. Note that λ , the sorting we have adopted in the previous section, is a P, R -sorting. We will establish the following:

1. $S \mid I$ is R -polite and (P^-, R^+) -ready;
2. T_0^R is (P^+, R^-) -disciplined.

By the main partial confluence theorem, Theorem 4.1.26, and since P_0 is P, R, S -closed, by the sorting and Lemma 2.5.7, we have that $P_0 \simeq_1 Q_0$. Since the agents are fully convergent, $P_0 \simeq Q_0$, as required. It remains to prove the two claims. This is the purpose of the next two lemmas.

Lemma 5.2.4 $S \mid I$ is R -polite and (P^-, R^+) -ready.

PROOF: The fact that $S \mid I$ is R -polite follows directly from the definitions, where we may see that the only occurrence of a name $r \in R$ in object position is within a restricted prefix of the form $(\nu r)\bar{p}(\tilde{x}, r)$.

A derivative of $S \mid I$ is a parallel composition whose components are the replicators S and I as well as derivatives of them representing operations being executed in various states. The names of the sorts R are return links for questions via names of sorts P . The only occurrence of a name $r \in R$ in subject position is immediately under an output prefix of the form $(\nu r)\bar{p}(\tilde{x}, r)$, where $p : P$. It follows that any derivative of $S \mid I$ is of the form

$$Q \equiv (\nu \tilde{z})(\Sigma_{i \in I_1} r_1(\tilde{x}_{1i}). P_{1i} \mid \dots \mid \Sigma_{i \in I_n} r_n(\tilde{x}_{ni}). P_{ni} \mid Q'),$$

where $r_1 \dots r_n : R$ are pairwise distinct and no other name of sort R occurs free in a P_{ji} or in Q' . Thus it is straightforward to see that $S \mid I$ is an R -confluent agent.

Let $\{\mathcal{P}^{\tilde{r}} \mid \tilde{r} \text{ a finite subset of } R\}$ be the partition of the state space of $S \mid I$ such that $Q \in \mathcal{P}^{\tilde{r}}$ if \tilde{r} are the names of sorts R occurring free in Q . This partition can easily be shown to be (P^-, R^+) -tidy:

1. If $Q \in \mathcal{P}^{\tilde{r}}$ and $Q \xrightarrow{\alpha} Q'$, where $\text{subj}(\alpha) \notin P^- \cup R^+$ then from the definitions we see that α must be τ . It is also clear that the free names of Q are not affected as a result of an internal action and so $Q' \in \mathcal{P}^{\tilde{r}}$.
2. If $Q \in \mathcal{P}^{\tilde{r}}$ and $Q \xrightarrow{\alpha} Q'$, where $\alpha \in P^-$ then $\alpha = (\nu r)\overline{p} * \text{srch}\langle k, r \rangle$ or $\alpha = (\nu r)\overline{p} * \text{ins}\langle k, b, r \rangle$, where $r : R$. Since $r \notin \tilde{r}$ and r occurs free in Q' , $Q' \in \mathcal{P}^{\tilde{r}, r}$.
3. If $Q \in \mathcal{P}^{\tilde{r}}$ and $Q \xrightarrow{\alpha} Q'$, where $\alpha \in R^+$ then by the definitions it follows that r does not occur free in Q' . So $Q' \in \mathcal{P}^{\tilde{r}-r}$.

Moreover, if $Q \in \mathcal{P}^{\tilde{r}}$ then it is clear that for all $r \in \tilde{r}$, Q can perform any action with positive subject r . Hence Q is (P^-, R^+) -ready. \square

Since $S \mid I$ is only capable of P^- actions of the form $(\nu \tilde{y})\overline{p} * \overline{m}\langle \tilde{x}, r \rangle$ where $r \in \tilde{y}$, it is sufficient, by Theorem 4.1.26, to consider T_0^R . So we may prove the following.

Lemma 5.2.5 T_0^R is (P^+, R^-) -disciplined.

PROOF: A derivative of T_0^R is a parallel composition whose components are derivatives of node agents and the agent **STORE**. The names of the sorts R are return links for questions via names of sort P . It follows from the definitions that if a name of sorts R occurs free within a derivative T of T_0^R then it occurs in negative subject position either unguarded or under a prefix of the form $\overline{put}\langle p \rangle$ and there is at most one occurrence of the second kind:

$$T \equiv (\nu \tilde{z})(\overline{r_1}\langle x_1 \rangle. P_1 \mid \dots \mid \overline{r_n}\langle x_n \rangle. P_n \mid \overline{put}\langle p \rangle. \overline{r_0}. P \mid T')$$

where $\tilde{r} : R$ and no other name of sorts R occurs free in a P_i or in T' . It then follows that T_0^R is R -confluent.

Let $\{\mathcal{P}^{\tilde{r}} \mid \tilde{r} \text{ a finite subset of } R\}$ be the partition of the state space of T_0^R such that $T \in \mathcal{P}^{\tilde{r}}$ if \tilde{r} are the names of sort R occurring free in T . Using arguments similar to those in the previous lemma, this partition can be shown to be (P^+, R^-) -tidy.

Moreover, if $T \in \mathcal{P}^{\tilde{r}}$ then it is not difficult to see that either $T \xrightarrow{\beta}$ or if $T = (\nu \tilde{z})(\overline{put}\langle p \rangle. \overline{r}. P \mid Q)$ then $T \xrightarrow{\tau} T' = (\nu \tilde{z})(\overline{r}. P \mid Q') \xrightarrow{\beta}$ where $\text{subj}(\beta) = \tilde{r}$. Note that since the agents **STORE** and **ROOT**, the only agents bearing and handling

name *put* respectively, are S-confluent, by Lemma 4.1.6, $T \simeq T'$. Hence Q is (P^+, R^-) -disciplined. \square

Thus the proof of the theorem is completed by appealing to Theorem 4.1.26. \square

5.2.2 Analysis of Q_0

Since $P_0 \simeq Q_0$ we may now restrict our attention to agent Q_0 . In this section we present an analysis of its behaviour and identify properties and invariants crucial to the correctness of the operations. We begin with a definition.

Definition 5.2.6 Suppose T is a derivative of T_0 . The functions *range*, *access* and *path* are defined as follows:

- $\text{range}(\text{ROOT}\langle \dots \rangle) = \emptyset$
 $\text{range}(\text{NODE}\langle \dots \rangle) = \emptyset$
 $\text{range}(\text{LEAF}\langle p, \tilde{k}, \tilde{b}_q \rangle) = (k_1, k_n], \text{ if } \# \tilde{k} = n$
- $\text{access}(\text{ROOT}\langle \dots \rangle) = (-\infty, \infty)$
 $\text{access}(\text{NODE}\langle p, \tilde{k}, \tilde{p} \rangle) = (k_1, \infty)$
 $\text{access}(\text{LEAF}\langle p, \tilde{k}, \tilde{b}_q \rangle) = (k_1, \infty)$
- Let k, p be a pair such that $p : P_N$ and p is uniquely borne by a component P of T . Then

$$\text{path}(k, p, T) = \begin{cases} \text{nil}, & \text{if } k \notin \text{access}(P) \\ \langle p \rangle, & \text{if } k \in \text{range}(P) \\ p \text{ path}(k, p', T), & \text{otherwise} \end{cases}$$

where if $P = \text{NODE}\langle q, \tilde{k}, \tilde{p} \rangle$ then $p' = \text{find}_N(k, \tilde{p}, \tilde{k})$, otherwise, if $P = \text{LEAF}\langle q, \tilde{k}, \tilde{b}_q \rangle$ then $p' = \text{find}_L(k, \tilde{b}, \tilde{k})$.

The function *range* associates with a leaf node the range of keys that either are in the node or could be and the empty set to all non-leaf nodes. The function *access* associates with a node the interval (k, ∞) , where k is the minimum key of the node. Intuitively, this is the set of keys that are accessible from the node. Finally, the function *path* returns the set of pointers to the nodes visited during a search for a key k in tree T , starting from a node with name p .

The following definition prescribes a number of properties that a derivative of T_0 should satisfy.

Definition 5.2.7 Let

$$T \equiv (\nu \tilde{z})(\text{STORE}(\tilde{q}) \mid \text{ROOT}(p, \tilde{k}, \tilde{p}) \mid \Pi_{i \in I} \text{NODE}(p_i, \tilde{k}_i, \tilde{p}_i) \mid \Pi_{j \in J} \text{LEAF}(p_j, \tilde{k}_j, \tilde{b}_j q_j))$$

be a derivative of T_0 . Then T is a *legal tree* if the following hold:

1. \tilde{k} and \tilde{k}_m , for all m , are in strictly ascending order;
2. for all $p : P$, $p \in n(T)$, p is uniquely borne in T ;
3. last $\tilde{q} = p$ and for all i such that $q_i \neq p$, q_i is uniquely borne by some $\text{NODE}(p_m, \tilde{k}_m, \tilde{p}_m)$, where $\text{hd } \tilde{k}_m = -\infty$ and $\text{hd } \tilde{p}_m = q_{i-1}$ if $i > 1$, and $\text{hd } \tilde{p}_m$ is uniquely borne by a component LEAF of T , otherwise;
4. if $m, n \in J$ and $m < n$ then $k_{mi} < k_{nj}$ for all i, j , except $k_{ml} = k_{(m+1)1}$ where $l = |\tilde{k}_m|$;
5. for all p, k , if p is borne by component P of T and $k \in \text{access}(P)$ then $\text{path}(k, p, T) = p_1 \dots p_m$ where if p_i is borne by component N_i then $k \in \text{access}(N_i)$ and $k \in \text{range}(N_m)$.

Conditions 1 and 2 give some straightforward properties satisfied by a B -tree, namely that the keys of each node are in ascending order and that each node has a distinct name. Condition 3 expresses a correctness property concerning the STORE of a tree: it ensures that it contains pointers to the leftmost nodes in order of creation, that is beginning from the lowest non-leaf level to the top. Condition 4 requires that the ranges of two different leaves are disjoint. Finally, Condition 5 states that if key k is accessible from node N with name p of a legal tree T then $\text{path}(k, p, T)$ is a finite sequence of pointers concluding at the leaf where k should exist. This implies that a search for k from node N will be successful. This property is crucial to the correctness of the operations. We have the following lemma:

Lemma 5.2.8 Let Q be a derivative of Q_0 . Then $Q \Rightarrow B$ where B is of the form

$$(\nu \tilde{z})(S\langle s, \text{get} \rangle \mid I\langle i, \text{get}, \text{que} \rangle \mid \Pi_{i \in I} \text{Search}\langle y_i, p_i, a_i \rangle \mid \Pi_{j \in J} \text{Down}\langle y_j, b_j, a_j, p_j, \tilde{p}_j \rangle \\ \mid \Pi_{k \in K} \text{Add}\langle y_k, b_k, a_k, p_k, \tilde{p}_k \rangle \mid \Pi_{l \in L} \text{Up}\langle y_l, q_l, p_l, \tilde{p}_l \rangle \mid \Pi_{m \in M} \overline{a_m} \langle \tilde{x}_m \rangle \mid T^b)$$

where I, J, K, L, M are disjoint and the following hold:

1. $\text{fn}(B) = \{s, i, a_n \mid n \in I \cup J \cup K \cup L \cup M\} \cup \text{fn}(T) \upharpoonright B$;
2. T is a legal tree;

3. for all $n \in I \cup J \cup K$, p_n is borne by a component C of T such that $y_n \in \text{access}(C)$;
4. if $n \in L$ then p_n is borne by a component C of T such that $y_n = \min \text{access}(C)$;
5. for all $p \in \{p_l \mid l \in L\} \cup \bigcup_{n \in J \cup K \cup L} \widetilde{p}_n$, if p is borne by some component C of T , $C \neq \text{LEAF}$.

This lemma asserts some invariants of the state space of Q_0 . According to the first property the free names of a derivative of Q_0 are names of sort B, corresponding to pointers to records in the database, the names i and s via which searches and insertions may be invoked and names of sort A via which the answers to all initiated operations are to be returned. Moreover, Q is either in a 'stable' form as defined by B or it may enter such a state after a number of internal actions that do not change its state. Such a stable process contains the following components: a legal tree and a number of agents corresponding to the search and insert operations currently in progress, in various states. An important property, Clause 3 above, is that any search or insertion regarding a key k and currently questioning a node N is such that k is accessible from node N . Since T is a legal tree we may then deduce that the operation may be completed successfully. Informally, the truth of this property is due to the fact that although data may be moved within the tree, this may only take place as a result of a split of a node and in a left to right direction. Thus, link pointers may be followed in order to reach the required data.

The last two properties concern derivatives of the insert agent during the Up -phase of the insertion. According to Property 4, for every pair (y, q) to be inserted by an Up process, there exists a node with name q and lowest key y . An observation used in proving this property is that the minimum key of a node remains constant during its lifetime. Finally, Property 5 states that such agents do not operate at the leaf level.

PROOF: The proof is by induction on the length of the derivation of Q . Although long it is mostly routine apart from the points where the definition of a 'legal tree' is appealed to. For the base case the claim clearly holds. So suppose $Q_0 \Rightarrow Q' \xrightarrow{\alpha} Q$ where Q' has the properties stated in the lemma. First suppose $Q' \equiv B$ for some B as above. Several possibilities exist:

1. $\alpha = s\langle k, a \rangle$ and $Q' \xrightarrow{\alpha} Q = (\nu \tilde{z})(\text{get}(p). \text{Search}\langle k, p, a \rangle \mid S \mid \dots \mid T^b)$. By the induction hypothesis, T is a legal tree, so suppose:

$$T = (\nu \tilde{p})(\text{STORE}(\tilde{q}, \text{get}, \text{put}, \text{que}) \mid \text{ROOT}\langle p, \dots \rangle \mid \dots)$$

where $\text{last } \tilde{q} = p$. By definition of STORE, $T \xrightarrow{\text{get}(p)} T$. Hence,

$$Q \xrightarrow{\tau} Q'' = (\nu \tilde{z}) \text{Search}\langle k, p, a \rangle \mid \dots \mid T^b.$$

By definition, $\text{access}(\text{ROOT}) = (-\infty, \infty)$, so $k \in \text{access}(\text{ROOT})$ and the result follows.

2. $\alpha = i\langle k, b, a \rangle$. This is similar to the previous case.
3. $\alpha = \tau$ and $Q' \xrightarrow{\tau} Q = (\nu \tilde{z})(S' \mid \dots \mid T^b)$ where for some $i \in I$,

$$\begin{aligned} \text{Search}\langle y_i, p_i, a_i \rangle & \xrightarrow{(\nu r) \overline{p_i \text{search}\langle y_i, r \rangle}} S' = r(\text{done}, q). \overline{a_i}\langle q \rangle. 0, \\ & + r(p') \text{Search}\langle y_i, p', a \rangle \\ & + r(\text{link}, p'). \text{Search}\langle y_i, p', a \rangle \end{aligned}$$

and

$$T = (\nu \tilde{p})(N \mid \dots) \xrightarrow{p_i \text{search}\langle y_i, r \rangle} T' = (\nu \tilde{p})(N' \mid \dots).$$

Suppose $p_i : P$ and $N = \text{LEAF}\langle p, \tilde{k}, \tilde{b}q \rangle$. The case $N = \text{NODE}$ is similar. There exist two possibilities:

- if $y_i \leq \text{last } \tilde{k}$ then $N' \xrightarrow{\tilde{r}\langle \text{done}, b \rangle} N\langle p_i, \tilde{k}, \tilde{b}q \rangle$,
- if $y_i > \text{last } \tilde{k}$ then $N' \xrightarrow{\tilde{r}\langle \text{link}, q \rangle} N$,

where $b = \text{find}_L(y_i, \tilde{b}, \tilde{k})$. In the former case

$$Q \xrightarrow{\tau} R = (\nu \tilde{z})(\overline{a_i}\langle p \rangle. 0 \mid \dots \mid T^b).$$

Since R is of the form required this completes the proof. In cases the latter case

$$Q \xrightarrow{\tau} (\nu \tilde{z})(\text{Search}\langle y_i, q, a_i \rangle \mid \dots \mid T^b).$$

By the induction hypothesis, $y_i \in \text{access}(N\langle p_i, \tilde{k}, \tilde{p} \rangle)$ and $\text{path}(y_i, p_i, T) = q_1 \dots q_n$, where q_i is uniquely borne by some component N_i of T and $y_i \in \text{access}(N_i)$. By definition of path, $p_i = q_1$ and $p = q_2$. So, p is uniquely borne by component N_2 of T and $y_i \in \text{access}(N_2)$. Hence, Property 3 holds and the remainder of the claim follows by the induction hypothesis.

4. $\alpha = \tau$ and $Q' \xrightarrow{\tau} Q = (\nu \tilde{z})(D' \mid \dots \mid T^b)$, where for some $j \in J$,

$$\text{Down}\langle y_j, b_j, a_j, p_j, \tilde{p}_j \rangle \xrightarrow{(\nu r) \overline{p_j \text{search}\langle y_j, r \rangle}} D' \text{ and } T \xrightarrow{p_j \text{search}\langle y_j, r \rangle} T'.$$

This is similar to the previous case.

5. $\alpha = \tau$ and $Q' \xrightarrow{\tau} Q = (\nu \tilde{z})(U' \mid \dots \mid T^b)$ where, for some $l \in L$

$$Up\langle y_l, q_l, p_l, \tilde{p}_l \rangle \xrightarrow{(\nu r)\overline{p_l \text{ ins}}\langle y_l, q_l, r \rangle} U' \text{ and } T \xrightarrow{p_l \text{ ins}\langle y_l, q_l, r \rangle} T'.$$

Various possibilities exist corresponding to the cases where the node participating in the communication is full or non-full, a leaf, an internal node or the root and depending on whether the insertion is to be made in that node. We consider two of the most interesting.

- Suppose $T = (\nu \tilde{p})(\text{NODE}\langle p_l, \tilde{k}, \tilde{p} \rangle \mid \dots)$ where, $y_l < \text{last } \tilde{k}$ and the node is full, that is $\#\tilde{k} = 2m + 1$. Then

$$\begin{aligned} T' &= (\nu \tilde{p})((\nu p')\bar{r}\langle p', h \rangle. (\text{NODE}\langle p_l, \tilde{k}_1, \tilde{p}_1 \rangle \mid \text{NODE}\langle p', \tilde{k}_2, \tilde{p}_2 \rangle) \mid \dots) \\ &\xrightarrow{\bar{r}\langle p', h \rangle} T'' = (\nu \tilde{p})(\text{NODE}\langle p_l, \tilde{k}_1, \tilde{p}_1 \rangle \mid \text{NODE}\langle p', \tilde{k}_2, \tilde{p}_2 \rangle \mid \dots) \end{aligned}$$

where $h = \min \tilde{k}_2 = \max \tilde{k}_1$. So $Q \xrightarrow{\tau} (\nu \tilde{z})(Up\langle h, p', \text{hd } \tilde{p}_l, \text{tl } \tilde{p}_l \rangle \mid \dots \mid T''^b)$. By construction Clauses 1-4 in the definition of a legal tree are satisfied by T'' , Clause 3 due to the fact that $\text{hd } \tilde{k} = \text{hd } \tilde{k}_1$. Moreover, suppose $k \in \text{access}(\text{NODE}\langle p', \tilde{k}_2, \tilde{p}_2 \rangle)$. Then $\text{path}(k, p', T'') = \text{path}(k, p_l, T)$. On the other hand, if $k \in \text{access}(\text{NODE}\langle p, \tilde{k}_1, \tilde{k}_2 \rangle)$ then it is easy to see that

$$\begin{aligned} \text{path}(k, p, T'') &= \text{path}(k, p, T), \text{ if } k \leq h \text{ and} \\ &= p' \text{ path}(k, p, T), \text{ otherwise.} \end{aligned}$$

Hence, Property 5 in the definition of a legal tree is satisfied. By the induction hypothesis, properties 1 and 3 of the theorem hold; note that $\min \tilde{k}_l = \min \tilde{k}_1$ so $\text{access}(\text{NODE}\langle p_l, \tilde{k}, \tilde{p} \rangle) = \text{access}(\text{NODE}\langle p_l, \tilde{k}_1, \tilde{p}_1 \rangle)$. Finally, since $h = \min \tilde{k}_2$, Property 4 also follows.

- Suppose $T = (\nu \tilde{p})(\text{ROOT}\langle p_l, \tilde{k}, \tilde{p} \rangle \mid \dots)$ where the node is full, that is $\#\tilde{k} = 2m + 1$. Then

$$\begin{aligned} T' &= (\nu \tilde{p})((\nu p_0, p')\overline{put}\langle p_0 \rangle. \bar{r}. (\text{NODE}\langle p_l, \tilde{k}_1, \tilde{p}_1 \rangle \mid \text{NODE}\langle p', \tilde{k}_2, \tilde{p}_2 \rangle) \\ &\quad \mid \text{ROOT}\langle p_0, \langle -\infty, h, \infty \rangle, \langle p_l, p, \text{nil} \rangle, put \rangle) \\ &\quad \mid \text{STORE}\langle \tilde{r}, get, put, que \rangle \mid \dots). \end{aligned}$$

T' may engage in the following transitions:

$$\begin{aligned} T' &\xrightarrow{\tau} T_1 = (\nu \tilde{p})((\nu p_0, p')\bar{r}. (\text{NODE}\langle p_l, \tilde{k}_1, \tilde{p}_1 \rangle \mid \text{NODE}\langle p', \tilde{k}_2, \tilde{p}_2 \rangle) \\ &\quad \mid \text{ROOT}\langle p_0, \langle -\infty, h, \infty \rangle, \langle p_l, p, \text{nil} \rangle, put \rangle) \\ &\quad \mid \text{STORE}\langle \tilde{r}p_0, get, put, que \rangle \mid \dots) \end{aligned}$$

$$\begin{aligned} \xrightarrow{\bar{r}} T_2 = & (\nu \tilde{p})((\nu p_0, p')(\text{NODE}\langle p_l, \tilde{k}_1, \tilde{p}_1 \rangle \mid \text{NODE}\langle p', \tilde{k}_2, \tilde{p}_2 \rangle \\ & \mid \text{ROOT}\langle p_0, \langle -\infty, h, \infty \rangle, \langle p_l, p, \text{nil} \rangle, \text{put} \rangle \\ & \mid \text{STORE}\langle \tilde{r}, \text{get}, \text{put}, \text{que} \rangle) \mid \dots) \end{aligned}$$

Hence, $Q \xrightarrow{\tau} (0 \mid \dots \mid T_2^b)$. Using arguments similar to those in the previous case it can be shown that the desired properties hold. In particular consider $\text{path}(k, p_0, T_2)$. By definition, $\text{path}(k, p_0, T_2) = p_0 \text{path}(k, p_l, T_2)$, if $k \leq h$ and $p_0 \text{path}(k, p', T_2)$, otherwise. By construction, $\text{path}(k, p_l, T_2) = \text{path}(k, p_l, T)$ and $\text{path}(k, p', T_2) = \text{path}(k, p_l, T)$. Hence, by the induction hypothesis, Clause 5 in the definition of a legal tree is satisfied.

6. $\alpha = \tau$ and $Q' \xrightarrow{\tau} Q = (\nu \tilde{z})(A' \mid \dots \mid T')$ where for some $k \in K$,

$$\text{Add}\langle y_k, b_k, a_k, p_k, \tilde{p}_k \rangle \xrightarrow{(\nu r) p_k^{*ins}\langle y_k, b_k, r \rangle} A' \text{ and } T \xrightarrow{p_k^{*ins}\langle y_k, b_k, r \rangle} T'.$$

This is similar to the previous case.

It remains to consider the case $Q' \xrightarrow{\alpha} Q$ and $Q' \Rightarrow B$ for some $B = (\nu \tilde{z})(S \mid I \mid A \mid T^b)$ as set out in the lemma. It can be observed from the analysis above that Q' has the following form: $Q' = (\nu \tilde{z})(S \mid I \mid A' \mid T^b)$ where $T' \in \mathcal{T}^r$ for some $r : R$. Moreover, let Q'' be such that $Q' \xrightarrow{\tau} Q'' \Rightarrow B$. If $\alpha = s\langle k, a \rangle$ then $Q = (\nu \tilde{z})(S \mid I \mid \text{get}(q). \text{Search}\langle k, q, a \rangle \mid A' \mid T^b)$ and clearly

$$\begin{aligned} Q & \Rightarrow (\nu \tilde{z})(S \mid I \mid \text{get}(q). \text{Search}\langle k, q, a \rangle \mid A \mid T^b) \\ & \xrightarrow{\tau} B' = (\nu \tilde{z})(S \mid I \mid \text{Search}\langle k, q, a \rangle \mid A \mid T^b) \end{aligned}$$

where B' is of the form required. The proof is similar for $\alpha = i\langle k, b, a \rangle$.

Finally, if $\alpha = \tau$ then since $T \in \mathcal{T}^r$, $T^b \not\xrightarrow{\beta}$ for any β such that $\text{subj}(\beta) \in P$. Furthermore, close inspection of the agent yields that $Q = Q''$ (both the tree and the body are capable of exactly one action). As a result, $Q \Rightarrow B$ and the result follows. This completes the proof. \square

Definition 5.2.9 Suppose

$$T \equiv (\nu \tilde{z})(\text{STORE} \mid \text{ROOT}\langle p, \tilde{k}, \tilde{p}, \text{put} \rangle \mid \prod_{i \in I} \text{NODE}\langle p_i, \tilde{k}_i, \tilde{p}_i \rangle \mid \prod_{j \in J} \text{LEAF}\langle p_j, \tilde{k}_j, \tilde{b}_j, q_j \rangle)$$

is a derivative of T_0 . The function *contents* is defined as follows:

$$\text{contents}(T, k) = \begin{cases} b_{mn}, & \text{if } k_{mn} = k \text{ for some } m \in J \\ \text{nil}, & \text{otherwise} \end{cases}$$

Using the previous lemma we may prove the following significant result. It states that any initiated search or insertion may terminate successfully and produce the appropriate result while having the required effect on the contents of the tree.

Lemma 5.2.10 Suppose Q is a derivative of Q_0 . The following hold:

- If $Q = (\nu\tilde{z})(C \mid \text{Search}\langle k, p, a \rangle \mid T^b)$ then $Q \Rightarrow Q' \xrightarrow{\bar{a}(b)} (\nu\tilde{z})(C \mid T^b)$ where $\text{contents}(T, k) = b$, and
- If $Q = (\nu\tilde{z})(C \mid \text{Insert}\langle k, b, a, p, \tilde{p} \rangle \mid T^b)$ where $\text{Insert} = \text{Down}$ or $\text{Insert} = \text{Add}$ then $Q \Rightarrow Q' \xrightarrow{\bar{a}} (\nu\tilde{z})(C \mid T^b)$ where $\text{contents}(T', y) = \text{contents}(T, y)$, if $y \neq k$ and b otherwise.

PROOF: The proof follows by appealing to Clause 5 in the definition of a legal tree and properties 2 and 3 of Lemma 5.2.8. Let us consider the second part of the lemma. The proof of the first part is easier.

Suppose $Q = (\nu\tilde{z})(C \mid \text{Down}\langle k, b, a, p, \tilde{p} \rangle \mid T^b)$ where Q is a derivative of Q_0 . By Clause 3 of Lemma 5.2.8, p is uniquely borne by some component Q of T such that $k \in \text{access}(Q)$. Moreover, by the same lemma, T is a legal tree. Hence, $\text{path}(k, p, T) = p_1 \dots p_m$ where p_i is uniquely borne by some component N_i of T with $k \in \text{access}(N_i)$ and $k \in \text{range}(N_m)$. By the definition of path and the definitions of the agents it is easy to see that

$$\begin{aligned} Q &\Rightarrow (\nu\tilde{z})(C \mid \text{Down}\langle k, b, a, p_1, \tilde{p}_1 \rangle \mid T^b) \\ &\xrightarrow{\tau} \dots \\ &\xrightarrow{\tau} Q' = (\nu\tilde{z})(C \mid \text{Add}\langle k, b, a, p_k, \tilde{p}_k \rangle \mid T^b). \end{aligned}$$

Two possibilities exist depending on whether N_k is full or not. Suppose that is is not full. Then

$$Q' \xrightarrow{\tau} \xrightarrow{\tau} (\nu\tilde{z})(C \mid \bar{a}.0 \mid T^b)$$

where, if $T = (\nu\tilde{p})(N_k \mid R)$ and $N_k = \text{LEAF}\langle p_k, \tilde{k}, \tilde{b}_q \rangle$, then $T' = (\nu\tilde{p})(\text{LEAF}\langle p_k, \tilde{k}', \tilde{b}'_q \rangle \mid R)$ and $\tilde{k}' = k_1 \dots k_l k k_{l+1} \dots k_l$, $\tilde{b}' = b_1 \dots b_l q b_{l+1} \dots b_n$, if $k_l < k < k_{l+1}$, and $\tilde{k}' = \tilde{k}$, $\tilde{b}' = b_1 \dots b_{l-1} b b_{l+1} \dots b_n$, if $k = k_l$. The case of N_k being full is similar. In either case, $\text{contents}(T', y) = \text{contents}(T, y)$, if $y \neq k$ and b otherwise. \square

Using this lemma we can prove that two derivatives of Q_0 involving trees that possess the same contents and processing the same operations but possibly at different stages (ignoring those responsible for balancing the tree after insertions) are in fact branching bisimilar.

Lemma 5.2.11 Suppose

$$\begin{aligned} Q_1 = & (\nu\tilde{z})(S \mid I \mid \prod_{i \in I} \text{Search}\langle k_i, p_i, a_i \rangle \mid \prod_{j \in J} B_j \langle k_j, b_j, a_j, p_j, \tilde{p}_j \rangle \mid \prod_{k \in K} \bar{a}_k \langle \tilde{x}_k \rangle \\ & \mid \prod U p \langle \dots \rangle \mid T_1^b) \end{aligned}$$

and

$$Q_2 = (\nu \tilde{z})(S \mid I \mid \Pi_{i \in I} \text{Search}\langle k_i, p'_i, a_i \rangle \mid \Pi_{j \in J} B'_j\langle k_j, b_j, a_j, p'_j, \tilde{p}'_j \rangle \mid \Pi_{k \in K} \overline{a_k}\langle \tilde{x}_k \rangle \mid \Pi Up\langle \dots \rangle \mid T_2^b)$$

are derivatives of Q_0 where $\text{contents}(T_1, k) = \text{contents}(T_2, k)$ for all k and either $B_j = B'_j = \text{Down}$ or $B_j = \text{Down}$ and $B'_j = \text{Add}$. Then $Q_1 \simeq Q_2$.

PROOF: The proof considers the relation

$$\mathcal{R} = \{(Q_1, Q_2) \mid Q_1, Q_2 \text{ are as above}\}$$

and it shows that this is a branching bisimulation by appealing to Lemma 5.2.10. It is mostly routine and the details are omitted. \square

This result provides interesting insights into the execution of the operations and has some further consequences. First, it formalizes the intuition that Up agents do not change the state of the system up to branching bisimilarity. In addition, as a result, the presence of the procedure responsible for reorganizing the tree is irrelevant to the functional correctness of the operations and could be omitted. In fact we may consider two additional insert operations, one where the agent Up is omitted and another where the return of the result (via name a) takes place only after agent Up completes execution. It can be verified (by considering some further simple invariants satisfied by the system) that the systems equipped with each of these operations are indistinguishable from each other.

The next lemma proves some properties of Q_0 that will be useful in proving that Q_0 is a (Q^+, A^-) -server system.

Lemma 5.2.12 Let Q be a derivative of Q_0 . Then

1. If Q is A^- -inert and $Q \xrightarrow{\tau} Q'$, then either $Q \simeq Q'$ or $Q' \xrightarrow{\alpha} Q''$ for some $\alpha \in A^-$ and Q'' is A^- -inert.
2. If $a \in \text{fn}(Q)[A$ then either $Q \xrightarrow{\bar{a}(\tilde{u})} \simeq$ or $Q \xrightarrow{\tau} \simeq \xrightarrow{\bar{a}(\tilde{u})} \simeq$ for some \tilde{u} .
3. If $\text{fn}(Q)[A = \emptyset$ and $Q \xrightarrow{u} \simeq \xrightarrow{\gamma} \simeq$ where $u \in Q^{+*}$, $\gamma \in \{\tau\} \cup A^-$, then there exists $\beta \in u$ such that $Q \xrightarrow{\beta} \simeq \xrightarrow{\gamma} \simeq$.
4. If $a \in \text{fn}(Q)[A$, where a is uniquely handled in Q and $Q \xrightarrow{\tau} \simeq Q_1 \xrightarrow{\bar{a}(\tilde{u})} \simeq$, $Q \xrightarrow{\tau} \simeq Q_2 \xrightarrow{\bar{a}(\tilde{u})} \simeq$, then $Q_1 \simeq Q_2$.

PROOF: The first property expresses that each decisive τ -action results in a new answer becoming available. Its proof is a case analysis on $Q \xrightarrow{\tau} Q'$. Suppose

$Q \equiv (\nu \tilde{z})(C \mid T^b)$, where C is a derivative of $S \mid I$ and T^b is a derivative of T_0^b . The following possibilities exist:

1. $Q' \equiv (\nu \tilde{z}')(C' \mid T^b)$, where $C \xrightarrow{\rho} C'$, $T^b \xrightarrow{\bar{\rho}} T^b$, $\rho \text{ comp } \bar{\rho}$ and $\text{subj}(\rho) : R$. By R-confluence, $Q \simeq Q'$ which completes the case.
2. $Q' \equiv (\nu \tilde{z})(C \mid T^b)$, where $T^b \xrightarrow{\tau} T^b$. This corresponds to the action performed by the tree structure when a root is split and the name of the new root is communicated to the STORE process via name *put*. Since components STORE and ROOT are S-confluent and no other component of the system owns name *out*, by Lemma 4.1.6, $Q \simeq Q'$.
3. $Q' \equiv (\nu \tilde{z}')(C' \mid T^b)$, where $C \xrightarrow{\rho} C'$, $T^b \xrightarrow{\bar{\rho}} T^b$, $\rho \text{ comp } \bar{\rho}$ and $\text{subj}(\rho) : P$. Various possibilities exist:

- $Q \equiv (\nu \tilde{z})(C \mid \text{Search}\langle k, p, a \rangle \mid T^b) \xrightarrow{\tau} Q' \equiv (\nu \tilde{z})(C \mid S' \mid T^b)$. It follows from the definition of *Search* that S' may evolve in two possible ways: either $Q' \xrightarrow{\tau} Q_1 \equiv (\nu \tilde{z})(C \mid \text{Search}\langle k, p', a \rangle \mid T^b)$ or $Q' \xrightarrow{\tau} Q_2 \equiv (\nu \tilde{z})(C \mid \bar{a}\langle b \rangle.0 \mid T^b)$. In the former case, by R-confluence $Q' \simeq Q_1$, and by Lemma 5.2.11, $Q \simeq Q_1$. Hence $Q \simeq Q'$. In the latter case, $Q_2 \xrightarrow{\bar{a}\langle b \rangle} Q'_2$ and clearly Q'_2 is A^- -inert. Hence Property 1 is satisfied.
- $Q \equiv (\nu \tilde{z})(C \mid \text{Down}\langle k, b, a, p, \tilde{p} \rangle \mid T^b) \xrightarrow{\tau} Q' \equiv (\nu \tilde{z})(C \mid D' \mid T^b)$. From the definition of *Down*, D' may evolve in two possible ways: either $Q' \xrightarrow{\tau} Q_1 \equiv (\nu \tilde{z})(C \mid \text{Down}\langle k, b, a, p', \tilde{p}' \rangle \mid T^b)$ or $Q' \xrightarrow{\tau} Q_2 \equiv (\nu \tilde{z})(C \mid \text{Add}\langle k, b, a, p, \tilde{p} \rangle \mid T^b)$. In either case, by R-confluence $Q' \simeq Q_i$, and by Lemma 5.2.11, $Q \simeq Q_i$. Hence $Q \simeq Q'$.
- $Q \equiv (\nu \tilde{z})(C \mid \text{Add}\langle k, b, a, p, \tilde{p} \rangle \mid T^b) \xrightarrow{\tau} Q' \equiv (\nu \tilde{z})(C \mid A' \mid T^b)$. It follows from the definition of *Add* that A' may evolve in two possible ways: either $Q' \xrightarrow{\tau} Q_1 \equiv (\nu \tilde{z})(C \mid \text{Add}\langle k, b, a, p', \tilde{p}' \rangle \mid T^b)$ or $Q' \xrightarrow{\tau} Q_2 \equiv (\nu \tilde{z})(C \mid \bar{a}.A \mid T^b)$ for some A . In the former case, by R-confluence $Q' \simeq Q_1$, and by Lemma 5.2.11, $Q \simeq Q_1$. Hence $Q \simeq Q'$. In the latter case, $Q_2 \xrightarrow{\bar{a}} Q'_2$ and clearly, Q'_2 is A^- -inert. Hence Property 1 is satisfied.
- $Q \equiv (\nu \tilde{z})(C \mid \text{Up}\langle k, q, p, \tilde{p} \rangle \mid T^b) \xrightarrow{\tau} Q' \equiv (\nu \tilde{z})(C \mid U' \mid T^b)$. It follows from the definition of *Up* that U' may evolve in two possible ways: either $Q' \xrightarrow{\tau} Q_1 \equiv (\nu \tilde{z})(C \mid \text{Up}\langle k, q, p', \tilde{p}' \rangle \mid T^b)$, or $Q' \xrightarrow{\tau} Q_2 \equiv (\nu \tilde{z})(C \mid A \mid T^b)$, where A is 0 or $\text{Up}\langle k', q', p', \tilde{p}' \rangle$. In both cases, by Theorem 5.2.8, T' is a legal tree. Moreover, $\text{contents}(T, k) = \text{contents}(T', k)$ for all k . Hence by Lemma 5.2.11, $Q \simeq Q_i$. Since by R-confluence $Q' \simeq Q_i$, we have that $Q \simeq Q'$ which completes the case.

This completes the proof of Property 1. Property 2 captures the fact that if an answer to an outstanding question is not already available it may become available after a single decisive internal action. Its proof follows easily using arguments similar to Lemma 5.2.10 making additional use of the first property. On the other hand, Property 3 asserts that there exists a question that is uniquely responsible for each decisive action and each answer. The proof is routine and is omitted.

Finally we sketch the proof of Property 4. This says that determination of an answer results in a uniquely determined state. We consider the case $a : A_S$. The case $a : A_I$ is similar. First we note that since $a \in \text{fn}(Q)$, Q must have a component $\text{Search}\langle k, p, a \rangle$ for some k and p . Suppose

$$Q \Rightarrow R_1 \equiv (\nu \tilde{z})(C_1 \mid T_1^b) \xrightarrow{\tau} Q_1 \Rightarrow R'_1 \xrightarrow{\bar{a}\langle u \rangle}$$

and

$$Q \Rightarrow R_2 \equiv (\nu \tilde{z})(C_2 \mid T_2^b) \xrightarrow{\tau} Q_2 \Rightarrow R'_2 \xrightarrow{\bar{a}\langle u' \rangle}$$

where $Q \simeq R_1 \simeq R_2$ and $Q_1 \simeq R'_1$, $Q_2 \simeq R'_2$. Since $R_1 \simeq R_2$ it must be that the tree-agents T_1, T_2 have the same contents; otherwise it is easy to construct examples that give a contradiction to $R_1 \simeq R_2$. Moreover, as illustrated in the analysis above, C_1 and C_2 consist of the same significant components (that is non- Up agents) in progress.

Since a is uniquely handled in R_1 and R_2 , C_1 and C_2 must be of the form

$$C_1 \equiv C'_1 \mid \text{Search}\langle k, p_1, a \rangle$$

$$C_2 \equiv C'_2 \mid \text{Search}\langle k, p_2, a \rangle$$

Moreover, the subsequent τ -actions represent the questioning of the leaf where k resides if it exists. Hence

$$Q_1 \equiv (\nu \tilde{z}')(C'_1 \mid r(\text{done}, x). \bar{a}\langle x \rangle \mid T_1^b)$$

$$Q_2 \equiv (\nu \tilde{z}')(C'_2 \mid r(\text{done}, x). \bar{a}\langle x \rangle \mid T_2^b)$$

where $T_1^b \xrightarrow{\bar{r}\langle \text{done}, u \rangle} T_1^b$ and $T_2^b \xrightarrow{\bar{r}\langle \text{done}, u' \rangle} T_2^b$ and by Lemma 5.2.10, $\text{contents}(T_1, k) = u$ and $\text{contents}(T_2, k) = u'$. Since T_1 and T_2 have the same contents, $u = u'$. Moreover, by R-confluence,

$$Q_1 \simeq (\nu \tilde{z}')(C'_1 \mid \bar{a}\langle u \rangle \mid T_1^b)$$

$$Q_2 \simeq (\nu \tilde{z}')(C'_2 \mid \bar{a}\langle u \rangle \mid T_2^b)$$

and by Lemma 5.2.11, $Q_1 \simeq Q_2$. □

5.2.3 Main Results

We now return to the proof of Theorem 5.2.2. Since $P_0 \simeq Q_0$ we can restrict attention to the system Q_0 . Let \mathcal{Q} be the transition system generated by Q_0 and \mathcal{F} the transition system generated by F_0 . Let \mathbf{Q} be the sorts \mathbf{Q}_I and \mathbf{Q}_S of the names used by the environment for initiating insertions and searches, and \mathbf{A} the sorts \mathbf{A}_I and \mathbf{A}_S of names to be used by the system for returning the results of the operations to the environment. Note that λ , the sorting of Section 5.1, is a \mathbf{Q}, \mathbf{A} -sorting. We will establish the following:

1. $\mathcal{Q}^{\mathbf{A}}$ is a $(\mathbf{Q}^+, \mathbf{A}^-)$ -server.
2. $\mathcal{F}^{\mathbf{A}}$ is a $(\mathbf{Q}^+, \mathbf{A}^-)$ -server.
3. $Q_0^b \simeq F_0^b$, where Q_0^b and F_0^b are the states corresponding to Q_0 in $\mathcal{Q}^{\leq 1}$ and to F_0 in $\mathcal{F}^{\leq 1}$ respectively.

Thus, by the theorem on social confluence, Theorem 4.3.9, we have that for any \mathbf{Q}, \mathbf{A} -closed context $\mathcal{C}[\cdot] \stackrel{\text{def}}{=} (\nu \tilde{z})(E \mid \cdot)$, where E is an \mathbf{A} -polite, $(\mathbf{Q}^-, \mathbf{A}^+)$ -client, $\mathcal{C}[Q_0] \simeq \mathcal{C}[Q_0^b]$ and $\mathcal{C}[F_0^b] \simeq \mathcal{C}[F_0]$. Moreover, by Property 3 above, it follows that $\mathcal{C}[Q_0^b] \simeq \mathcal{C}[F_0^b]$. Hence we have

$$\mathcal{C}[Q_0] \simeq \mathcal{C}[Q_0^b] \simeq \mathcal{C}[F_0^b] \simeq \mathcal{C}[F_0]. \quad (5.1)$$

Note that the first branching bisimilarity, $\mathcal{C}[Q_0] \simeq \mathcal{C}[Q_0^b]$, is interesting in itself, as it stipulates that the system Q_0 is indistinguishable from its serial version in certain contexts. In order to complete the proof, we choose an appropriate, simple context. Let E_0 be the agent

$$E_0 \stackrel{\text{def}}{=} !in(k, p, r). (\nu a) \bar{i}(k, p, a). a. \bar{r}. 0 \\ \mid !sr(k, r). (\nu a) \bar{s}(k, a). a(b). \bar{r}(b). 0$$

where $in, sr : \mathbf{Q}' \neq \mathbf{Q}$ and $r : \mathbf{A}' \neq \mathbf{A}$. It is straightforward to see that E_0 is an \mathbf{A} -polite, $(\mathbf{Q}^-, \mathbf{A}^+)$ -client agent. Consider $(\nu i, s)(E_0 \mid Q_0)$. By the sorting discipline and Lemma 2.5.7, we may see that the agent is \mathbf{Q}, \mathbf{A} -closed. Then, by Equation 5.1 above, $(\nu i, s)(E_0 \mid Q_0) \simeq (\nu i, s)(E_0 \mid F_0)$. Using this property we will establish the required result. Let \mathcal{R} be the following relation:

$$\mathcal{R} = \{(Q, F) \mid Q_0 \xrightarrow{s} Q, F_0 \xrightarrow{s} F, \text{ for some } s, (\nu \tilde{z})(E \mid Q) \simeq (\nu \tilde{z})(E \mid F) \\ \text{where } E = E_0 \mid \prod_{a \in I} a(\tilde{x}). \bar{r}_a(\tilde{x}), I = \text{fn}(Q) \upharpoonright \mathbf{A}, \\ \text{and the } r_a \text{ are distinct}\}.$$

We will show that $\mathcal{R} \subseteq \simeq_A$. Since $(Q_0, F_0) \in \mathcal{R}$ it then follows that $Q_0 \simeq_A F_0$.

Suppose $(Q, F) \in \mathcal{R}$ and $E, S_1 = (\nu \tilde{z})(E \mid Q), S_2 = (\nu \tilde{z})(E \mid F)$ are as above. Suppose $Q \xrightarrow{\alpha} Q'$. The following possibilities exist:

- $\alpha = s\langle k, a \rangle$, where $a \notin \text{fn}(Q)$. By construction, $F \xrightarrow{\alpha} F'$. Moreover, with r a fresh name

$$\begin{array}{lcl} E & \xrightarrow{sr\langle k, r \rangle} & E' = E \mid (\nu a)\bar{s}\langle k, a \rangle. a(b). \bar{r}\langle b \rangle \\ & \xrightarrow{(\nu a)\bar{s}\langle k, a \rangle} & E'' = E \mid a(b). \bar{r}\langle b \rangle. \end{array}$$

Thus,

$$\begin{array}{lcl} S_1 = (\nu \tilde{z})(E \mid Q) & \xrightarrow{sr\langle k, r \rangle} & S_1'' = (\nu \tilde{z})(E' \mid Q) \xrightarrow{\tau} S_1' = (\nu \tilde{z}a)(E'' \mid Q') \\ S_2 = (\nu \tilde{z})(E \mid F) & \xrightarrow{sr\langle k, r \rangle} & S_2'' = (\nu \tilde{z})(E' \mid F) \xrightarrow{\tau} S_2' = (\nu \tilde{z}a)(E'' \mid F') \end{array}$$

where by Q-confluence and Lemma 4.1.6, $S_1' \simeq S_1''$, $S_2' \simeq S_2''$. Since $S_1 \simeq S_2$, and the agents can perform the actions $sr\langle k, r \rangle$ in only one way, $S_1'' \simeq S_2''$. Thus, $S_1' \simeq S_2'$. Hence, $F \xrightarrow{\alpha} F'$ and $(Q', F') \in \mathcal{R}$.

- $\alpha = i\langle k, b, a \rangle$, where $a \notin \text{fn}(Q)$. This is similar to the previous case.
- $\alpha \in A^-$. Suppose $\alpha = \bar{a}\langle \tilde{v} \rangle$. Then by the definition of \mathcal{R} , $E = a(\tilde{x}). \bar{r}\langle x \rangle. \mid C$ for some C and some $r : A'$ where r does not occur free in C . Hence $E \xrightarrow{a(\tilde{v})} E'$ and

$$S_1 \xrightarrow{\tau} S_1'' = (\nu \tilde{z})(E' \mid Q') \xrightarrow{\bar{r}\langle \tilde{v} \rangle} S_1' = (\nu \tilde{z})(C \mid Q').$$

Since $S_1 \simeq S_2$,

$$S_2 \Rightarrow S'' = (\nu \tilde{z}')(E'' \mid F'') \xrightarrow{\bar{r}\langle \tilde{v} \rangle} S' \simeq S_1'$$

where $S'' \simeq S_1''$. By A-confluence and Lemma 4.1.6, $S_1 \simeq S_1''$, so $S_1 \simeq S''$ and $S_2 \simeq S''$. We observe the following: Suppose $S_2 \xrightarrow{\tau} S = (\nu \tilde{z})(E_1 \mid F_1)$. Three possibilities exist.

- If $F \xrightarrow{\tau} F_1$ and $F \simeq F_1$ then it is clear that $S_2 \simeq S$.
- If $F \xrightarrow{\tau} F_1$ and $F \not\simeq F_1$, that is if F has performed a decisive action determining the result of one of the operations in progress, then it is easy to construct examples showing that $S_2 \not\simeq S$. For example, if the action corresponds to the determination of the value b associated with a key k and assuming that x is the name via which the result of the operation should be returned, then clearly, for all $s, x \notin \text{fn}(s)$,

$$F' \xRightarrow{s} \xrightarrow{\bar{x}\langle b \rangle}$$

whereas

$$F \xrightarrow{\text{in}(k,b',r)} \bar{x}(b').$$

Thus, $F \not\approx F'$.

- Otherwise, $F \xrightarrow{\beta} F_1$ and $E \xrightarrow{\bar{\beta}} E_1$ where $\text{subj}(\beta) : A$, $\beta \text{ comp } \bar{\beta}$ (note that by the construction of \mathcal{R} , E is not capable of immediately engaging in any actions in Q^\pm). By A-confluence and Lemma 4.1.6, $S_2 \simeq S$.

So consider $S_2 \Rightarrow S''$. Since $S_2 \simeq S''$, by the observation above there exists s, \bar{s} , $s \in A^{+*}$, $s \text{ comp } \bar{s}$ such that $E \xrightarrow{s} \simeq E''$ and $F \xrightarrow{\bar{s}} \simeq F''$. Moreover, by the construction of E and since r does not occur in C it must be that $E'' = \bar{\tau}(\tilde{v}) \mid C'$ for some C' . Hence $a(\tilde{v}) \in s$ and $\bar{a}(\tilde{v}) \in \bar{s}$. By the construction of F it follows that $F \xrightarrow{\bar{a}(\tilde{v})} F'$ and by A-confluence, $F' \xrightarrow{\bar{s}/\bar{a}(\tilde{v})} \simeq F''$ and $E' \xrightarrow{s/a(\tilde{v})} \simeq E''$. As a result

$$S_2 \xrightarrow{\tau} S_2' = (\nu \tilde{z})(E' \mid F') \xrightarrow{\bar{\tau}(\tilde{v})} S_2' = (\nu \tilde{z})(C \mid F').$$

By Lemma 4.1.6, $S_2' \simeq S_2 \simeq S''$ and so $S_2' \xrightarrow{\bar{\tau}(\tilde{v})} \simeq S'$. By the construction of \mathcal{R} , $S_2' \simeq S'$ and so $S_2' \simeq S_1'$. Thus, $F \xrightarrow{\alpha} F'$ and $(Q', F') \in \mathcal{R}$ as required.

- $\alpha = \tau$ and $Q \xrightarrow{\tau} Q'$. Then $S_1 \xrightarrow{\tau} S_1' = (\nu \tilde{z})(E \mid Q')$. If $S_1 \simeq S_1'$ then $(Q', F) \in \mathcal{R}$. Otherwise, as observed above, it must be that $Q \not\approx Q'$ and since $S_1 \simeq S_2$, $S_2 \Rightarrow \xrightarrow{\tau} S_2' \simeq S_1'$. Suppose

$$S_2 = (\nu \tilde{z})(E_1 \mid F_1) \xrightarrow{\tau} \dots \xrightarrow{\tau} S_{2n} = (\nu \tilde{z})(E_n \mid F_n) \xrightarrow{\tau} S_2' = (\nu \tilde{z})(E' \mid G)$$

Since $S_2 \simeq S_{2n}$, $E \xrightarrow{s} \simeq E_n$, $F \xrightarrow{\bar{s}} \simeq F_n$ where $s = \alpha_1 \dots \alpha_n$, $\alpha_i \in A^+$ and $s \text{ comp } \bar{s}$. On the other hand, $S_{2n} \not\approx S_2'$ so $E_n = E'$, $F_n \xrightarrow{\tau} G$.

Since $F \xrightarrow{\bar{s}} \simeq F_n \xrightarrow{\tau} G$ where $\bar{s} \in A^{-*}$, we may see by the structure of F that $F \xrightarrow{\tau} F' \xrightarrow{\bar{s}} \simeq G$. So $S_2 \xrightarrow{\tau} S_2' = (\nu \tilde{z})(E \mid F')$ and since additionally $E \xrightarrow{s} \simeq E_n$ and $F' \xrightarrow{\bar{s}} \simeq G$ by A⁻-confluence, $S_2' \simeq S_2'$. Thus $S_2' \simeq S_1'$. This implies that $F \xrightarrow{\tau} F'$ and $(Q', F') \in \mathcal{R}$ as required.

The converse is similar. This completes the proof. \square

The remainder of this section is concerned with proving the three claims above.

Theorem 5.2.13 Q_0^A is a (Q^+, A^-) -server.

PROOF: First we note that by the definition of Q_0^A we need only consider Q^+ actions in which new A-names are received.

Consider a derivative P of Q_0^A . As illustrated in the proof of Lemma 5.2.8, P contains two names of sort Q , namely s and i and these are uniquely owned by components S and I of P respectively. Further, they both occur unguarded in positive subject position within a replicator process. Neither of them occurs within a summation or a parallel composition. Hence it is easy to see that P is Q -confluent.

To see that P is also A -confluent is slightly more involved. The names of sort A are return links via which the answers of operations initiated via names of sort Q are to be returned. From the definitions it is apparent how these names are used and mentioned: a name $a : A$ may occur in agents of the form $Search\langle a \rangle$, $Down\langle a \rangle$ or $Add\langle a \rangle$ in a negative subject position. It does not occur in a guarded summation or within a parallel composition and it does not occur in the agent under the prefix in which it occurs. In fact P has the form

$$P \equiv (\nu \tilde{p})(\overline{a_1}\langle \dots \rangle. P_1 \mid \dots \mid \overline{a_n}\langle \dots \rangle. P_n \mid C)$$

where the $a_1 \dots a_n$ are pairwise distinct, and no free names of sort A occur free in a P_i or free and unguarded in C . Clearly, P is A -confluent. Furthermore, we may see that the following hold:

1. Suppose $P \xrightarrow{\alpha} P_1 \xrightarrow{\beta} P_2$ where either $\alpha \notin Q^\pm \cup \{\tau\}$ or $\beta \notin A^\pm \cup \{\tau\}$. It is easy to see by the definitions of Q_0 that $P \xrightarrow{\beta} P'_1 \xrightarrow{\alpha} P'_2 \simeq P_2$. For example, if $\alpha \in A^-$ and $\beta \in Q^+$ then in fact $P_2 \equiv P'_2$.
2. Suppose $P \xrightarrow{\alpha} P_1$ and $P \xrightarrow{\beta} P_2$ where $\alpha \in Q^+$, $\alpha \neq \beta$. Then it is easy to see that $P_2 \xrightarrow{\alpha}$ as all derivatives of P_0 are Q^+ -enabled.

Thus P is Q, A -socially confluent. To obtain a (Q^+, A^-) -server partition of Q_0^A we take $\{\mathcal{P}^{\tilde{a}} \mid \tilde{a} \text{ finite subset of } A\}$, where $P \in \mathcal{P}^{\tilde{a}}$ if \tilde{a} are the names of sort A occurring free in P . To check that this is a server partition we have:

- 3 If $P \in \mathcal{P}^{\tilde{a}}$ and $P \xrightarrow{\alpha} P'$ where $\alpha \notin Q^+ \cup A^-$ then from the definitions we see that α must be τ . It is easy to check that the free names of P are not affected as a result of a τ action and so $P' \in \mathcal{P}^{\tilde{a}}$.
- 4 If $P \in \mathcal{P}^{\tilde{a}}$ and $P \xrightarrow{\alpha} P'$ where $\alpha \in Q^+$ and $\alpha = s\langle k, a \rangle$ or $\alpha = i\langle k, b, a \rangle$ where a is a fresh name, then since $a \notin \tilde{a}$ and a occurs free in a *Search* or *Down* process as the result of the invocation, $P' \in \mathcal{P}^{\tilde{a}, a}$.
- 5 If $P \in \mathcal{P}^{\tilde{a}}$ and $P \xrightarrow{\alpha} P'$ where $\alpha \in A^-$ then certainly $a = \text{subj}(\alpha)$ must occur free in P . Moreover, by the definitions it follows that a does not occur free within P' . So $P \in \mathcal{P}^{\tilde{a}-a}$.

Finally, by Lemma 5.2.12, it follows that Q_0^A is a (Q^+, A^-) -server system. \square

Theorem 5.2.14 F_0^A is a (Q^+, A^-) -server.

The proof of this is straightforward and is omitted. \square

Theorem 5.2.15 $Q_0^b \simeq F_0^b$.

PROOF: It is not difficult to see that since Q_0^b and F_0^b are obtained by pruning the transition systems of Q_0 and F_0 so that at most one operation is active at any point they are in fact branching bisimilar. An observation that plays an important rôle in the proof is that, as stated in Lemma 5.2.10, Q_0 is guaranteed to answer questions ‘correctly’. This can be proved rigorously but the argument is routine and is omitted. \square

5.2.4 Full Results

In this section we discuss the proof of Theorem 5.2.1, asserting the full correctness criterion of the operations. The proof is similar to the proof of Theorem 5.2.2 and makes use of previous results concerning the behaviour of the systems in question (such as Lemma 5.2.12) but it is based on the social confluence theory of Section 4.4 and thus requires attention to the fact that different operations may provide the same name for the return of an answer.

Since $P_0 \simeq Q_0$ we can restrict attention to the system Q_0 and we aim to prove that $Q_0 \simeq F_0$. Recall that \mathcal{Q} is the transition system generated by Q_0 and \mathcal{F} the transition system generated by F_0 . We will establish the following:

1. \mathcal{Q} is a $(Q^+, A^-)^\delta$ -server.
2. \mathcal{F} is a $(Q^+, A^-)^\delta$ -server.

By the main theorem of Section 4.4, Theorem 4.4.10, and Properties 1, 2 above we have that for any Q, A -closed context, $C[\cdot] \stackrel{\text{def}}{=} (\nu \tilde{z})(E \mid \cdot)$, where E is a $(Q^-, A^+)^\delta$ -client, $C[Q_0] \simeq C[Q_0^b]$ and $C[F_0^b] \simeq C[F_0]$. Moreover, by Theorem 5.2.15, $Q_0^b \simeq F_0^b$ and so $C[Q_0^b] \simeq C[F_0^b]$. Hence we have

$$C[Q_0] \simeq C[Q_0^b] \simeq C[F_0^b] \simeq C[F_0]. \quad (5.2)$$

In order to complete the proof, we choose an appropriate context. Let E_0 be the following agent.

$$\begin{aligned}
 E_0\langle \varepsilon \rangle &\stackrel{\text{def}}{=} \text{in}(k, p, r). (\nu a) \bar{i}\langle k, p, a \rangle. (E_0\langle a \rangle \mid a. \bar{r}. 0) \\
 &\quad + sr(k, r). (\nu a) \bar{s}\langle k, a \rangle. (E_0\langle a \rangle \mid a(b). \bar{r}\langle b \rangle. 0) \\
 E_0\langle t \rangle &\stackrel{\text{def}}{=} \text{in}(k, p, r). (\Sigma_{a \in t} \bar{i}\langle k, p, a \rangle. (E_0\langle t \rangle \mid a. \bar{r}. 0) \\
 &\quad + (\nu a) \bar{i}\langle k, p, a \rangle. (E_0\langle ta \rangle \mid a. \bar{r}. 0)) \\
 &\quad + sr(k, r). (\Sigma_{a \in t} \bar{s}\langle k, a \rangle. (E_0\langle t \rangle \mid a(b). \bar{r}\langle b \rangle. 0) \\
 &\quad + (\nu a) \bar{s}\langle k, a \rangle. (E_0\langle ta \rangle \mid a(b). \bar{r}\langle b \rangle. 0))
 \end{aligned}$$

Note that in $E_0\langle t \rangle$, t contains all A names that have been provided in operation invocations to be used for the return of results. Thus on invoking a new operation $E_0\langle t \rangle$ may provide either one of the names in t , or a new name.

It is straightforward to see that E_0 is a $(Q^+, A^-)^\delta$ -client agent. Then, by 5.2 above, $(\nu i, s)(E_0 \mid Q_0) \simeq (\nu i, s)(E_0 \mid F_0)$. Using this property we will establish the required result. Let \mathcal{R} be the following relation:

$$\begin{aligned}
 \mathcal{R} = \{ & (Q, F) \mid Q_0 \xrightarrow{s} Q, F_0 \xrightarrow{s} F, \text{ for some } s, (\nu \tilde{z})(E \mid Q) \simeq (\nu \tilde{z})(E \mid F) \\
 & \text{where } E \equiv E_0\langle t \rangle \mid \Pi_{a \in w} a(\tilde{x}). \bar{r}_a\langle \tilde{x} \rangle, a \in w \text{ iff } a \in \text{fn}(Q)[A, \\
 & |w|_a = n \text{ iff } n \text{ occurs in } Q, n \text{ times,} \\
 & w \subseteq t, \text{ and the } r_a \text{ are distinct} \}
 \end{aligned}$$

We will show that $\mathcal{R} \subseteq \simeq$. Since $(Q_0, F_0) \in \mathcal{R}$ it then follows that $Q_0 \simeq F_0$.

Suppose $(Q, F) \in \mathcal{R}$ and E is as above. Suppose $Q \xrightarrow{\alpha} Q'$. There exist various possibilities. Consider the following:

$\alpha = s\langle k, a \rangle$, where $a \in \text{fn}(Q)$. By construction, $F \xrightarrow{\alpha} F'$. Moreover, since $a \in \text{fn}(Q)$, for t such that $a \in t$, and r a fresh name

$$\begin{aligned}
 E \equiv E_0\langle t \rangle \mid C &\xrightarrow{sr\langle k, r \rangle} E' \equiv (\Sigma_{a \in t} \bar{s}\langle k, a \rangle. (E_0\langle t \rangle \mid a(b). \bar{r}\langle b \rangle) \\
 &\quad + (\nu a) \bar{s}\langle k, a \rangle (E_0\langle ta \rangle \mid a(b). \bar{r}\langle b \rangle) \mid C) \\
 &\xrightarrow{\bar{s}\langle k, a \rangle} E'' \equiv E_0\langle ta \rangle \mid a(b). \bar{r}\langle b \rangle \mid C
 \end{aligned}$$

Hence

$$\begin{aligned}
 S_1 \equiv (\nu \tilde{z})(E \mid Q) &\xrightarrow{sr\langle k, r \rangle} S'_1 \equiv (\nu \tilde{z})(E' \mid Q) \xrightarrow{\tau} S'_1 \equiv (\nu \tilde{z})(E'' \mid Q') \\
 S_2 \equiv (\nu \tilde{z})(E \mid F) &\xrightarrow{sr\langle k, r \rangle} S'_2 \equiv (\nu \tilde{z})(E' \mid F) \xrightarrow{\tau} S'_2 \equiv (\nu \tilde{z})(E'' \mid F')
 \end{aligned}$$

where by Q -confluence and Lemma 4.1.6, $S'_1 \simeq S'_1$, $S'_2 \simeq S'_2$. Since $S_1 \simeq S_2$, $S'_1 \simeq S'_2$ and so $S'_1 \simeq S'_2$. Hence $(Q', F') \in \mathcal{R}$.

The remaining cases are similar to those of the proof of Theorem 5.2.1. This completes the proof. \square

The remainder of this section is concerned with proving the two claims above.

Theorem 5.2.16 Q_0 is a $(Q^+, A^-)^\delta$ -server.

PROOF: As argued previously, Q_0 is Q -confluent. Since additionally no derivative of Q_0 may perform any Q^- actions, it is also Q^δ -confluent. To show that Q_0 is also A^δ -confluent is slightly more involved. Recall that names of sort A are return links via which the answers of operations initiated via names of sort Q are to be returned. From the definitions it is apparent that a name $a : A$ may occur in agents of the form $Search\langle a \rangle$, $Down\langle a \rangle$ or $Add\langle a \rangle$ in a negative subject position. In fact a derivative P of Q_0 has the form

$$P \equiv (\nu \tilde{p})(\overline{a_1}\langle \dots \rangle. P_1 \mid \dots \mid \overline{a_n}\langle \dots \rangle. P_n \mid C)$$

where no free names of sort A occur free in a P_i or free and unguarded in C . Note that in this setting, $a_1 \dots a_n$ need not be distinct. Nonetheless, it is easy to see that P is A^δ -confluent: although P may not be determinate with respect to A^- actions, for all $\alpha_1, \alpha_2 \in A^-$, if $P \xrightarrow{\alpha_1} P_1$, $P \xrightarrow{\alpha_2} P_2$, and $P_1 \not\approx P_2$ then $P_1 \xrightarrow{\alpha_2} P'$ and $P_2 \xrightarrow{\alpha_1} P'$. By (1) and (2) in the proof of Theorem 5.2.13 we deduce that Q_0 is $(Q^+, A^-)^\delta$ -socially confluent.

To obtain a $(Q^+, A^-)^\delta$ -server partition of Q_0 we take $\{\mathcal{P}^{\tilde{a}} \mid \tilde{a} \in A^*\}$ where $P \in \mathcal{P}^{\tilde{a}}$ if \tilde{a} are the names of sort A occurring free in P and, if $b \in \text{fn}(P) \setminus A$ occurs n times in P , then $|\tilde{a}|_b = n$. To check that this partition is $(Q^+, A^-)^\delta$ -tidy we have:

1. If $P \in \mathcal{P}^{\tilde{a}}$ and $P \xrightarrow{\alpha} P'$ where $\text{subj}(\alpha) \notin Q^+ \cup A^-$ then from the definitions we see that α must be τ . It is easy to check that the free names of P are not affected as a result of a τ action and so $P' \in \mathcal{P}^{\tilde{a}}$.
2. If $P \in \mathcal{P}^{\tilde{a}}$ and $P \xrightarrow{\alpha} P'$ where $\alpha \in Q^+$ and $\alpha = s\langle k, a \rangle$ or $\alpha = i\langle k, b, a \rangle$ where a is a fresh name, then since $a \notin \tilde{a}$ and a occurs free in a *Search* or *Down* process as the result of the invocation, $P' \in \mathcal{P}^{\tilde{a}, a}$.
3. If $P \in \mathcal{P}^{\tilde{a}}$ and $P \xrightarrow{\alpha} P'$ where $\alpha \in A^-$ then certainly $a = \text{subj}(\alpha)$ must occur free in P . Moreover, by the definitions it follows that a does not occur free within P' . So $P \in \mathcal{P}^{\tilde{a}-a}$.

Thus Q_0 is a $(Q^+, A^-)^\delta$ -base. Finally, by Lemma 5.2.12, it follows that Q_0 is a $(Q^+, A^-)^\delta$ -server. \square

Theorem 5.2.17 F_0 is a $(Q^+, A^-)^\delta$ -server.

The proof of this is straightforward and is omitted. \square

5.3 The Concurrent Search Structure

As mentioned earlier, the analysis of the simplified structure (with sequential nodes) considered in the previous section may be used to deduce the correctness of the operations when the nodes allow concurrent accesses. In this section we give the definitions of nodes that allow multiple reads and a single write to take place concurrently and verify the required results. The definitions of the concurrent nodes are similar to those of the sequential nodes. The sorting employed is supplemented by the sort U , where $\lambda(U) = \{(\text{int}^k, P_N^k), (\text{signal}, \text{int}^k, P_N^k) \mid 1 \leq k \leq 2m + 1\}$. The use of names of these sorts is discussed below.

The definition of the agent $\text{NODE}' \equiv \text{NODE}\langle p, \tilde{k}, \tilde{p} \rangle$ representing a concurrent node with name p , storing keys \tilde{k} and pointers \tilde{p} is given as follows:

$$\begin{aligned}
 \text{NODE}' &\stackrel{\text{def}}{=} (\nu u)(\text{RD}_N\langle u \rangle \mid \text{WR}_N\langle u \rangle) \\
 \text{RD}_N &\stackrel{\text{def}}{=} (u) (u(\tilde{k}', \tilde{p}'). \text{RD}_N\langle p, \tilde{k}', \tilde{p}' \rangle, u) \\
 &\quad + p * \text{srch}(k, r). (\text{RD}_N \mid \text{cond}(k > \text{last } \tilde{k} \triangleright \bar{r}\langle \text{link}, \text{find}_N(k, \tilde{p}, \tilde{k}) \rangle. 0, \\
 &\quad \quad \quad k \leq \text{last } \tilde{k} \triangleright \bar{r}\langle \text{find}_N(k, \tilde{p}, \tilde{k}) \rangle. 0))) \\
 \text{WR}_N &\stackrel{\text{def}}{=} (u) p * \text{ins}(k, q, r). \text{cond}(k > \text{last } \tilde{k} \triangleright \bar{r}\langle \text{find}_N(k, \tilde{p}, \tilde{k}) \rangle. \text{WR}_N, \\
 &\quad \quad \quad k \in \tilde{k} \quad \triangleright \bar{r}. \bar{u}\langle \tilde{k}, \tilde{p}'' \rangle. \text{WR}_N\langle p, \tilde{k}, \tilde{p}'' \rangle, \\
 &\quad \quad \quad \text{notfull} \quad \triangleright \bar{r}. \bar{u}\langle \tilde{k}', \tilde{p}' \rangle. \text{WR}_N\langle p, \tilde{k}', \tilde{p}' \rangle, \\
 &\quad \quad \quad \text{full} \quad \triangleright (\nu p')(\bar{r}\langle p', h \rangle. \bar{u}\langle \tilde{k}_1, \tilde{p}_1 \rangle. \\
 &\quad \quad \quad (\text{WR}_N\langle p, \tilde{k}_1, \tilde{p}_1 \rangle \mid \text{NODE}'\langle p', \tilde{k}_2, \tilde{p}_2 \rangle))) .
 \end{aligned}$$

where $\tilde{p}', \tilde{p}'', \tilde{p}_1, \dots$ are the same as in the sequential case. Thus a node is defined as a parallel composition of two components RD_N and WR_N , responsible for handling searches and insertions respectively. The private name $u : U$ is to be used by the WR_N component to inform RD_N of any changes to the contents of the node due to insertions. The RD component in its quiescent state may receive a message via name u and thus update its contents. Alternatively, it may receive a search request via name $p * \text{srch}$. It then assumes the state $\text{RD}_N \mid \text{cond}(\dots)$ in which it may receive new search operations and process the initial invocation (as recorded in the second component). Hence, multiple searches may be active within the node concurrently. On the other hand the WR operator allows at most one insertion at any time. Insertions are handled in the expected manner with the additional feature that if an insertion succeeds and the contents of the node are updated then the RD component of the node is informed via name u .

The definition of the agent $\text{LEAF}' \equiv \text{LEAF}'\langle p, \tilde{k}, \tilde{b} \rangle$ representing a leaf with name p , keys \tilde{k} and database pointers \tilde{b} is very similar, using the function find_L and indicating the end of a search with the keyword *done*. It is given as follows:

$$\begin{aligned}
\text{LEAF}' &\stackrel{\text{def}}{=} (\nu u) (\text{RD}_L\langle u \rangle \mid \text{WR}_L\langle u \rangle) \\
\text{RD}_L &\stackrel{\text{def}}{=} (u) (u\langle \tilde{k}', \tilde{b}' \rangle. \text{RD}_L\langle p, \tilde{k}', \tilde{b}', u \rangle \\
&\quad + p * \text{srch}(k, r). (\text{RD}_L \mid \text{cond} (k > \text{last } \tilde{k} \triangleright \bar{r}\langle \text{link}, \text{find}_L(k, \tilde{p}, \tilde{k}) \rangle. 0, \\
&\quad \quad \quad k \leq \text{last } \tilde{k} \triangleright \bar{r}\langle \text{done}, \text{find}_L(k, \tilde{b}, \tilde{k}) \rangle. 0))) \\
\text{WR}_L &\stackrel{\text{def}}{=} (u) p * \text{ins}(k, b, r). \text{cond} (k > \text{last } \tilde{k} \triangleright \bar{r}\langle \text{find}_L(k, \tilde{b}, \tilde{k}) \rangle. \text{WR}_L, \\
&\quad \quad \quad k \in \tilde{k} \triangleright \bar{r}. \bar{u}\langle \tilde{k}, \tilde{b}'' \rangle. \text{WR}_L\langle p, \tilde{k}, \tilde{b}'' \rangle, \\
&\quad \quad \quad \text{notfull} \triangleright \bar{r}. \bar{u}\langle \tilde{k}', \tilde{b}' \rangle. \text{WR}_L\langle p, \tilde{k}', \tilde{b}' \rangle, \\
&\quad \quad \quad \text{full} \triangleright (\nu q) (\bar{r}\langle q, h \rangle. \bar{u}\langle \tilde{k}_1, \tilde{b}_1 \rangle. \\
&\quad \quad \quad (\text{WR}_L\langle p, \tilde{k}_1, \tilde{b}_1 \rangle \mid \text{LEAF}'\langle q, \tilde{k}_2, \tilde{b}_2 \rangle))) .
\end{aligned}$$

Finally, the definition of the agent $\text{ROOT}' \equiv \text{ROOT}'\langle p, \tilde{k}, \tilde{p}, \text{put} \rangle$, representing a root node with name p , keys \tilde{k} , pointers \tilde{p} and name put to be used when the root is split is given as follows:

$$\begin{aligned}
\text{ROOT}' &\stackrel{\text{def}}{=} (\nu u) (\text{RD}_R\langle u \rangle \mid \text{WR}_R) \\
\text{RD}_R &\stackrel{\text{def}}{=} (u) (u\langle \tilde{k}', \tilde{p}' \rangle. \text{RD}_R\langle p, \tilde{k}', \tilde{p}', u \rangle \\
&\quad + u(\text{node}, \tilde{k}', \tilde{p}'). \text{RD}_N\langle p, \tilde{k}', \tilde{p}', u \rangle \\
&\quad + p * \text{srch}(k, r). (\text{RD}_R \mid \bar{r}\langle \text{find}_N(k, \tilde{p}, \tilde{k}) \rangle. 0) \\
\text{WR}_R &\stackrel{\text{def}}{=} (u) p * \text{ins}(k, p, r). \text{cond} (k \in \tilde{k} \triangleright \bar{r}. \bar{u}\langle \tilde{k}, \tilde{p}'' \rangle. \text{WR}_R\langle p, \tilde{k}, \tilde{p}'' \rangle, \\
&\quad \quad \quad \text{notfull} \triangleright \bar{r}. \bar{u}\langle \tilde{k}', \tilde{p}' \rangle. \text{WR}_R\langle p, \tilde{k}', \tilde{p}' \rangle, \\
&\quad \quad \quad \text{full} \triangleright (\nu p, p') (\bar{\text{put}}\langle p \rangle. \bar{r}. \bar{u}\langle \text{node}, \tilde{k}_1, \tilde{p}_1 \rangle. \\
&\quad \quad \quad (\text{WR}_N\langle p, \tilde{k}_1, \tilde{p}_1 \rangle \\
&\quad \quad \quad \mid \text{NODE}'\langle p', \tilde{k}_2, \tilde{p}_2 \rangle \\
&\quad \quad \quad \mid \text{NEWROOT}'))))
\end{aligned}$$

where $\text{NEWROOT}' \equiv \text{ROOT}'\langle p, \langle -\infty, h, \infty \rangle, \langle p, p', \text{nil} \rangle, \text{put} \rangle$. The last option in the conditional of the WR_R agent corresponds to the case where an insertion is to take place in a full root. As previously this results in the split of the root which assumes node status and the creation of a new root. In order to achieve this change of status the WR_R component of the root becomes WR_N . Moreover, a message communicating this change is sent to RD_R via name $u : \mathbf{U}$ including the signal node : signal. As captured in the definition of RD_R , receipt of this message results in the change of status of RD_R to RD_N .

The concurrent B^{link} -tree in its initial state is represented by the agent

$$\begin{aligned}
T'_0 &\stackrel{\text{def}}{=} (\nu p, p', \text{put}) (\text{STORE}(\langle p \rangle, \text{get}, \text{put}, \text{que}) \\
&\quad \mid \text{ROOT}'\langle p, \langle -\infty, \infty \rangle, \langle p', \text{nil} \rangle, \text{put} \rangle \mid \text{LEAF}'\langle p', \langle \infty \rangle, \langle \text{nil} \rangle \rangle)
\end{aligned}$$

and the system consisting of the concurrent structure and the operations is given as

follows:

$$P'_0 \stackrel{\text{def}}{=} (\nu get, que)(S \mid I \mid T'_0)$$

We now discuss the correctness of the operations in the concurrent search structure. The result asserting the correctness criterion is the following:

Theorem 5.3.1 $P'_0 \simeq F_0$.

In order to prove this result it suffices to show that $P_0 \simeq P'_0$. Then, since $P_0 \simeq F_0$, by Theorem 5.2.1 the result follows. The proof of $P_0 \simeq P'_0$ uses the theory of partial confluence. For P and R the sorts defined in the previous section we have the following:

Lemma 5.3.2 T_0^R is (P^+, R^-) -disciplined.

PROOF: The proof of this claim uses arguments identical to those in the proof of Lemma 5.2.5. \square

Lemma 5.3.3 $T_0^b \simeq T_0^{rb}$.

PROOF: It is not difficult to see that since T_0^b and T_0^{rb} are obtained by pruning the transition systems of T_0 and T'_0 respectively so that at most one operation is active at any time then they are in fact branching bisimilar. The proof is routine and the details are omitted. \square

Theorem 5.3.4 $P_0 \simeq P'_0$.

PROOF: By Lemma 5.2.3,

$$P_0 \simeq Q \equiv (\nu get, que)(S \mid I \mid T_0^b).$$

Moreover, by Lemma 5.2.4, Lemma 5.3.2 and since P_0 is a P, R -closed agent, by the main partial confluence theorem, Theorem 4.1.26, we have

$$P'_0 \simeq_1 Q' \equiv (\nu get, que)(S \mid I \mid T_0^{rb})$$

and since the agents are fully convergent, $P_0 \simeq Q'$. In addition by Lemma 5.3.3, $T_0^b \simeq T_0^{rb}$ and so $Q \simeq Q'$. Therefore $P_0 \simeq P'_0$ which completes the proof. \square

5.4 Deletion and compression

In this section we consider the deletion and compression algorithms of [Sag86] and propose an improvement to the latter. We then discuss how our analysis may be extended to establish the correctness of these algorithms. First we note that the

definitions of *NODE*, *ROOT* and *LEAF* must be augmented to support the additional operations. In particular, the process-calculus representation of a pointer to a node becomes a value of the record type $\{ins \vdash P_I, srch \vdash P_S, del \vdash P_D, com \vdash P_C\}$ so that if p is a node name, $p * del$ and $p * com$ are names of sorts P_D and P_C via which deletions and compressions may be initiated.

According to the algorithm in [Sag86], a deletion is handled by locating the leaf where the key to be deleted is and then removing the key and the pointer associated with it. If a deletion results in a leaf becoming less than half full then a compression process is employed to redistribute data so that each non-root node of the tree has at least $m + 1$ pairs. To illustrate the main idea of the compression algorithm suppose node A is less than half full. The compression process is responsible for locating a neighbouring sibling of A , say B . If A and B together have fewer than $2m + 1$ pairs then all their data is moved into one of the nodes and the other is deleted. Otherwise, data is moved from B to A so that each has at least $m + 1$ pairs. In either case the compression process is responsible for making a change in the parent of A . In the former case the pair associated with the deleted node needs to be removed whereas in the latter case the change of the range of A needs to be communicated.

Various subtle points become apparent when the algorithm is examined carefully. First, the possibility of deletion of the lowest and greatest key of a node requires that information concerning these is explicitly stored. In [Sag86], each node must store both of these keys. However, in the algorithm we propose a node need store only its greatest key. This is explained in greater detail below. More interesting is the issue of deletion of nodes. According to the compression algorithm a node may be deleted. Nonetheless, it is possible that other processes have a pointer to this node and may attempt to read it after its deletion. Hence a node cannot be removed altogether. Instead, when a node N is to be deleted its data is replaced by a single pointer pointing to the node where the search should continue. Informally this is the pointer to the node where the data of N was moved before its deletion. The definition of the agent $DELETED \equiv DELETED\langle h, p, q \rangle$ representing a deleted node with key h , name p and link pointer q is as follows:

$$\begin{aligned} DELETED \stackrel{\text{def}}{=} & p * srch(k, r). \bar{r}\langle q \rangle. DELETED + p * ins(k, p', r). \bar{r}\langle q \rangle. DELETED \\ & + p * del(k, r). \bar{r}\langle q \rangle. DELETED + p * com(k, p', r). \bar{r}\langle q \rangle. DELETED. \end{aligned}$$

In its quiescent state a deleted node may accept requests for all the operations. In response it returns the stored pointer. The key h represents the minimum key accessible from the deleted node. The reason for including it in the definition, is to

facilitate the proof of correctness by allowing us to define $\text{access}(\text{DELETED}\langle h, p, q \rangle) = (h, \infty)$.

We now discuss the delete and compress operations of [Sag86] in greater detail and present parts of their process-calculus descriptions. Then we propose an alternative compression algorithm and discuss how the previous analysis has influenced its design. Finally, we describe briefly how the existing proofs can be extended to prove the new algorithm's correctness.

5.4.1 The Algorithms

We begin with the process calculus description of the deletion operation [Sag86]: The agent $D \equiv D\langle d, \text{get} \rangle$ is defined as follows, where get is of type *signal*:

$$\begin{aligned}
 D &\stackrel{\text{def}}{=} !d(k, a). \text{get}(q). \text{Delete}\langle q, k, a, \varepsilon \rangle \\
 \text{Delete} &\stackrel{\text{def}}{=} (q \ k \ a \ \tilde{p}) (\nu r) \overline{q} * \text{srch}\langle k, r \rangle. (r(q'). \text{Delete}\langle q', k, a, q\tilde{p} \rangle \\
 &\quad + r(\text{link}, q'). \text{Delete}\langle q', k, a, \tilde{p} \rangle \\
 &\quad + r(\text{done}, b). \text{Del}\langle q, k, a, \tilde{p} \rangle) \\
 \text{Del} &\stackrel{\text{def}}{=} (q \ k \ a \ \tilde{p}) (\nu r) \overline{q} * \text{del}\langle k, r \rangle. (r(q'). \text{Del}\langle q', k, a, \tilde{p} \rangle \\
 &\quad + r. \bar{a}. 0 \\
 &\quad + r(\text{empty}, k'). \bar{a}. \bar{c}\langle q, k', \tilde{p} \rangle. 0).
 \end{aligned}$$

The agent D may repeatedly generate deletion processes when supplied via name d of sort *D* with a key k to delete and a name a via which to signal completion of the deletion. Its behaviour is similar to that of the inserter agent I , of Section 5.1. *Delete* follows a path through the tree, recording the rightmost node visited on each level, and when the appropriate leaf is found it requests the deletion of the appropriate key using the selector *del*. If the deletion results in the leaf becoming less than half full, the deletion process is informed of this by the *LEAF* agent in question sending it the value *empty* and a key. In such a case a compression process is activated via name c of a sort *C* to redistribute the leaf's data or delete it if it has become empty; this may lead to activation of other compression processes.

When the compression agent $C \equiv C\langle c \rangle$ below is supplied with the pointer to the half empty node, one of its keys, and the path recorded by the deletion process, it activates a compressor:

$$C \stackrel{\text{def}}{=} !c(q, k, \tilde{p}). \text{Compress}\langle q, k, \tilde{p} \rangle.$$

Although sharing properties with the *Up* phase of the insertion operation, *Compress* has a much more complicated and subtle behaviour. Informally, it behaves as follows: if the node is the root then *Compress* terminates, (the root is permitted to be less

than half empty). Otherwise, using the path received, *Compress* first locates and locks the parent P of the node A to be compressed. It then locks and examines node A . If A is no longer less than half full, due to insertions and compressions having taken place since its creation, *Compress* terminates. Similarly, if A is the rightmost child of P , the two nodes are unlocked and *Compress* terminates. Otherwise, the third node to be locked is A 's right neighbour, B . If P does not have a pointer to B then it must be that this is still to be inserted in P by an inserter in its *Up* phase. In this case all three nodes are unlocked and *Compress* repeats this part of its activity, beginning by locking P . It is expected that in the meantime a pointer to B will have been added to P and *Compress* will be able to proceed. If, on the other hand, P does have a pointer to B , then one of the following takes place:

1. If A and B have together no more than $2m$ pairs, the data of B is shifted into A , node B is deleted, and the old high key of A and the pointer to B are deleted from P . Then the nodes are unlocked. It is possible that due to the deletion of a pair from P , it may become less than half full. If either A or P is less than half full then further compression processes are initiated to rebalance the tree.
2. If A and B together have more than $2m$ pairs then pairs are shifted from B to A so that each has at least m pairs. The high key of A is updated in P and the three nodes are unlocked.

This process may be repeated in several levels of the tree and may reach the root. As with insertion agent *Up*, it is possible that the path provided by the deletion process may become empty although a compression is required at a higher level of the tree. If this happens, the compression process queries the *STORE* to obtain the names of the leftmost nodes at each of the new levels.

The algorithm has the following defect. Consider the compression of a node A described above and suppose that case (2) applies so that data of B is shifted into its left neighbour, A . Suppose that the compression is running concurrently with a process searching for a key k which is about to examine node B . Suppose additionally that key k belongs to the data shifted into A as a result of the compression. It is easy to see from the definition of the search operation that this fact will not be noticed. The search will continue in the subtree of B and thus fail erroneously. This problem was noted in [Sag86] and a solution was proposed in which processes are aborted and restarted. This involves storing the low key of each node explicitly and modifying the search phase of *all* of the operations so that during the search in a node for a key k , it is checked whether k is greater than the node's low key. If

it is, the operation may proceed; otherwise it is aborted and must be restarted at a higher level.

The compression algorithm we propose is a variant of the algorithm above which avoids the problem described. In its design our intention was to ensure that, like *Up*, *Compress* affects neither the contents of the tree nor the accessibility of nodes. Hence its actions do not change the branching-bisimilarity state of the system. It was observed in the analysis that an invariant maintained by the search and insertion algorithms which is crucial to their correctness is that the minimum key of a node is not altered during the node's lifetime. This and the fact that data is moved only from left to right ensures that operations may always be completed successfully, if necessary by using link pointers. It can be seen that in fact a weaker invariant property suffices: that the minimum of a node is never increased. (Note that in the algorithm of [Sag86] just described, the low key of a node may increase.)

The new algorithm maintains this invariant during the compression process: when a node *A* is half empty *Compress* locates and locks its parent *P*. By reading *P* it obtains a pointer to the node *B* expected to be the left (as opposed to the right) neighbour of *A*; it then locks *B* and *A*. If *A* is no longer half empty, *Compress* releases the three nodes and terminates. If the link pointer of *B* does not point to *A* then all three nodes are unlocked and *Compress* repeats this part of its activity, beginning by locking *P*. Otherwise, *Compress* performs one of (1) and (2) described above.

It is easy to see that either *B* is deleted and all its data is moved into *A*, or data is moved from *B* to *A*. Both cases result in the decrease of *A*'s lowest key and the movement of data takes place from left to right. Note that in the former case *B* becomes *DELETED*(*h*, *p*, *q*) where *h* is the minimum key of *B*, *p* is its pointer and *q* is the pointer to *A*.

However, if *A* is the leftmost child of a node then this procedure cannot be applied, *A* having no left neighbour. Instead *A*'s right neighbour *C* is visited. If *A* and *C* together have fewer than $2m + 1$ keys, then *A* is deleted and its data is moved to *C*. Otherwise, *Compress* releases the three nodes and repeats this part of its activity, beginning by locking *P*. The algorithm does not move data from *C* to *A*: this may cause failure of a process that subsequently tries to read *C* expecting to find information that has been moved to *A*. Although this last scenario may result in *Compress* locking and unlocking the three nodes several times and delay the progress of other processes, we may expect that it is likely to arise infrequently due to the 'normal' movement of data from left to right.

Hence the new algorithm maintains the invariants necessary to guarantee its correctness and that of the other operations, and in contrast to [Sag86], none of the other operations must be changed, and the abortion and restarting of operations is avoided. Moreover, the new algorithm does not require the addition of the low key to the nodes of the data structure and is at least as efficient as the original algorithm. We move on to discuss the proof of its correctness.

5.4.2 Correctness

Let C_0 be the system composed from a tree in its initial state and the operations S , I , D and C . The correctness of the algorithms may then be expressed by comparison with an agent B which gives a succinct description of the intended observable behaviour of the system. Agent B is similar to agent F in Section 5.2 and it takes the following additional parameters: the name d via which deletions may be initiated; a sequence δ (the *deletions*) of pairs consisting of a key k to be deleted and a name a via which the termination of the deletion may be signalled; and a sequence δ^c (the *completed deletions*) of names a whose keys have been deleted but which have not been used to signal this. We define $B \equiv B\langle i, d, s, f, \iota, \iota^c, \sigma, \sigma^c, \delta, \delta^c \rangle$ as follows:

$$\begin{aligned}
 B \stackrel{\text{def}}{=} & i(k, b, a). B\langle \dots, \iota \langle k, b, a \rangle, \dots \rangle \\
 & + s(k, a). B\langle \dots, \sigma \langle k, a \rangle, \dots \rangle \\
 & + d(k, a). B\langle \dots, \delta \langle k, a \rangle, \dots \rangle \\
 & + \sum_{\langle k, b, a \rangle \in \iota} \tau. B\langle \dots, f[b/k], \iota - \langle k, b, a \rangle, \iota^c a, \dots \rangle \\
 & + \sum_{\langle k, a \rangle \in \sigma} \tau. B\langle \dots, \sigma - \langle k, a \rangle, \sigma^c \langle a, f(k) \rangle, \dots \rangle \\
 & + \sum_{\langle k, a \rangle \in \delta} \tau. B\langle \dots, f[\text{nil}/k], \dots, \delta - \langle k, a \rangle, \delta^c \langle a \rangle \rangle \\
 & + \sum_{\langle a \rangle \in \iota^c} \bar{a}. B\langle \dots, \iota^c - \langle a \rangle, \dots \rangle \\
 & + \sum_{\langle a, b \rangle \in \sigma^c} \bar{a} \langle b \rangle. B\langle \dots, \sigma^c - \langle a, b \rangle, \dots \rangle \\
 & + \sum_{\langle a \rangle \in \delta^c} \bar{a}. B\langle \dots, \delta^c - \langle a \rangle \rangle.
 \end{aligned}$$

The first three summands represent initiation of new operations, the next three invisible completion of outstanding operations (with appropriate update of the association in the case of insertion and deletion), and the last three signals of outcomes to the environment.

Let $B_0 \equiv B\langle i, d, s, \lambda k. \text{nil}, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon \rangle$. The theorem asserting the correctness of the algorithm is the following:

Theorem 5.4.1 $C_0 \simeq B_0$.

This theorem may be proved by extending the proof in Section 5.2 to accommodate the deletion and compression operations. The main reworking concerns

Section 5.2.2; the results of Sections 5.2.1 and 5.3.3 follow in an almost identical way. In particular, Lemma 5.2.8 needs to be extended to consider derivatives of the *Delete* and *Compress* processes. It is then possible to deduce a new version of Lemma 5.2.10 asserting that deletion operations may terminate successfully. Moreover, Lemma 5.2.11 may be extended to show that the compression operation does not change the branching-bisimilarity state of the system. The arguments are lengthy and require careful attention to detail, but the proof is not difficult. Due to its length it is omitted.

Chapter 6

Concurrent Objects

The main aim of this chapter is to enunciate and prove the soundness of transformation rules for programs of a concurrent-object language. Its starting point is the work of C B Jones [Jon93a, Jon96] on formal development of concurrent programs utilizing ideas from object-oriented programming. A central part of that development process is the use of transformations to increase the scope for concurrent activity within systems of objects prescribed by programs without altering their observable behaviours. The results of the chapter concern a specific concurrent-object programming language, a variant of the $\pi o\beta\lambda$ language [Jon93a] which in turn is derived from the POOL family [Ame89]. This small language is rich enough for the problems to be interesting and difficult and for the concepts and techniques to be illustrated to good effect. The concepts and techniques are, however, widely applicable.

There has been a substantial amount of work on semantics for the POOL family of languages. In addition to the translational semantics mentioned in the Introduction, other semantic techniques have also been investigated. In [ABKR86] an operational semantics was given to a member of the family. A reformulation of this semantics as a two-level transitional semantics was presented in [Wal95a] and a close correspondence between this and a semantics by translation to the π -calculus was established. Furthermore, in [ABKR89], a denotational semantics was provided based on metric spaces. As one would expect, considering the kind of language under study, the metric-space model is complicated and its suitability for reasoning about programs remains to be investigated. Work on the POOL family also includes [Boe91], where a proof system for reasoning about the correctness of POOL programs was presented.

In this chapter we build on work using calculi of mobile processes as semantic

bases for concurrent-object languages. In our view these models with changing structure are very well suited to giving natural and direct semantic definitions of such languages. This method of semantic definition has the additional benefit that the process-calculus theory may be used to reason rigorously about classes of systems and individual systems prescribed by programs.

We begin our analysis by studying the notion of determinacy within the language and we isolate syntactic conditions which guarantee that programs conforming to them are determinate. We seek syntactic conditions on programs which ensure that references to objects may not be shared. To achieve this we augment the language with an expression form which expresses a destructive read of a variable. This gives a means of expressing cleanly that when one object sends to a second object a reference to a third object, the first object relinquishes the reference to the third. It would be very awkward to express this effect without the additional expression form. The determinacy of the sublanguage generated by these conditions is proved on the basis of the translational semantics to the π_v -calculus. A proof based on an operational semantics for the language was also undertaken in [PW95].

This study enhanced significantly our understanding of the concurrent-object language and it gave clear directions for the enunciation of the transformation rules: generalizations of the conditions play a rôle in the rules we consider. Moreover the notions of confluence and partial confluence are central to the proofs of their correctness. As shown in [LW95a], partial confluence is useful in reasoning about classes of non-confluent systems in which interaction between possibly non-confluent components is of a certain disciplined kind. Here we employ the theory of partial confluence accommodating divergence. This turns out to be necessary as in considering the correctness of the transformation rules, the possibility of non-termination of method invocations must be taken into account.

An outline of the chapter follows. In the next section we introduce the programming language and we give an informal account of its semantics. A formal semantics by translation is then presented in Section 6.2. In Section 6.3 we study determinacy in the language and isolate syntactic conditions which guarantee it. This claim is formally proved on the basis of the semantic definition in Section 6.4. In Section 6.5 the transformations are introduced; the proof of their correctness is undertaken in Section 6.6.

6.1 The programming language

We begin with a description of the programming language, which is a variant of the $\pi o\beta\lambda$ language [Jon93a] in turn derived from the POOL family [Ame89], and an informal discussion of its semantics. The language is statically-typed with types **bool** (Booleans), **int** (integers), **unit** (the one-element type) and **ref**(A) for A a class name. A value of type **ref**(A) is a reference to an object of class A ; classes are explained below. In the abstract syntax definitions we use A to range over class names, m over method names, X, Y over variable names, f over constants and operators of the Boolean, integer and unit types, E over expressions, and S over commands. The expressions and commands are the well-typed phrases given as follows:

$$E ::= X \mid X^\dagger \mid \text{new}(A) \mid f(\tilde{E}) \mid E!m(\tilde{E}) \mid \text{input}$$

$$S ::= X := E \mid \text{output } E \mid \text{return } E \mid E!m(\tilde{E}) \mid \text{commit } E!m(\tilde{E}) \\ \mid S_1; S_2 \mid \text{if } E \text{ then } S_1 \text{ else } S_2.$$

Declarations are given as follows, where T ranges over types. First, variable declarations are given by

$$Vdec ::= \text{var } X_1 : T_1, \dots, X_p : T_p.$$

Then, method declarations are given by

$$Mdec ::= \text{method } m(\tilde{Y} : \tilde{T}) : T, Vdec, S$$

where \tilde{Y} of types \tilde{T} are the formal parameters, T is the result type, and S is the body of the method with $Vdec$ declaring variables local to it. Sequences of method declarations are given by

$$Mdecs ::= Mdec_1, \dots, Mdec_q$$

and class declarations by

$$Cdec ::= \text{class } A, Vdec, Mdecs.$$

Finally, program declarations are given by

$$Pdec ::= Cdec_1, \dots, Cdec_r, \text{trigger } E_0$$

where E_0 is of the form $\text{new}(A)!m(\tilde{K})$.

$Pdec$ above prescribes the possible computations of a system of concurrent objects each of which is an instance of one of the classes declared in it. The expression E_0 acts as a trigger to initiate computation by creating an object of one

of the classes A and invoking one of its methods m with parameters \widetilde{K} where each parameter is a constant or the expression $\text{new}(IO)$ (see below). Arbitrarily-many objects may be created during computation, and references to objects (and simple values) may be passed in interactions between objects. In this way highly mobile systems may be described. Each object of class A , as in $Cdec$ above, has private variables, as declared in $Vdec$. On creation, it assumes a quiescent state in which each of its private variables has the value nil (representing a reference to no object for variables of the ref types and the undefined value for variables of types bool , int and unit), and any one of its methods, as declared in $Mdec$ s may be invoked. When an object α invokes in an object β its method m , as in $Mdec$ above (with some parameters), the activity of α is suspended until the result of the invocation is returned to it. On invocation, β executes the body S of m . It may return a result to α by executing a return command. Alternatively, β may, via a commit command, delegate to another object γ the responsibility for returning a result to α , thereby freeing itself to continue with some other activity. On completing execution of the method body S , β resumes its quiescent state; only then may another method be invoked in β . Objects may enjoy concurrent activity as the return of a result or the delegation of the responsibility for returning a result need not be the last action in a method body.

An informal account of the meanings of expressions and commands follows. Evaluation of X involves reading the value of the variable X . That of X^\dagger is similar except that the value of X becomes nil when it is read. Evaluation of $\text{new}(A)$ results in the creation of an object of class A ; the value of the expression is a reference to that object. f ranges over constants (0 , true , nil etc.) and simple operators ($+$, $=$ etc.). The evaluation of $E!m(\widetilde{E})$ involves the evaluation of E and then the expressions in the tuple \widetilde{E} followed by the invocation of method m with parameters the values of \widetilde{E} in the object to which the value of E is a reference. The value of the expression $E!m(\widetilde{E})$ is the simple value or reference returned to the object as the result of the method invocation. Evaluation of input consumes an integer from the environment of the system, which becomes the value of the expression. The result type of a command of the form $E!m(\widetilde{E})$ is unit . The assignment, sequence and conditional commands are standard. Execution of $\text{output } E$ involves evaluation of E and the output of its (integer) value to the environment. Execution of $\text{return } E$ involves evaluation of the expression E and return of its value to the object ultimately responsible for the invocation of the method in which the statement occurs. The commit command is explained below via an example class definition.

Interaction with the environment is the function of objects of the following distinguished class whose definition is assumed, without explicit mention, to be present in every program declaration:

```
class IO
method In():int
    return input
method Out(X:int):unit
    return nil ; output(X)
```

We restrict attention to programs in which `new(IO)` does not occur except in the trigger of a program so that at most one object of class `IO` exists at any point during computation. In addition we require that `input` and `output(E)` occur only within the `IO` class. Note however that this does not impose any severe constraints on the system's interaction with the environment as many objects may share the reference to the `IO` object and thereby `input` and `output` integer values by invoking the appropriate methods.

We illustrate the language in the example class declaration below from [Jon93a].

```
class T
var K:int, V:ref(A), L:ref(T), R:ref(T)
method insert(X:int, W:ref(A)):unit
    return nil ;
    if K=nil then (K:=X ; V:=W ; L:=new(T) ; R:=new(T))
    else if X=K then V:=W
    else if X<K then L!insert(X,W)
        else R!insert(X,W)
method search(X:int):ref(A)
    if K=nil then return nil
    else if X=K then return V
    else if X<K then commit L!search(X)
        else commit R!search(X)
```

This class may be used to construct binary tree-structured symbol tables, an example of which is shown in Figure 6.1.

An object of this class represents a node which stores in its variables `K`, `V`, `L`, `R` an integer key, a value (a reference to an object of some class `A`), and references to two instances of the class (its left and right children in the tree structure of which it is a component). It has two actions: the method `insert` which allows a key-value

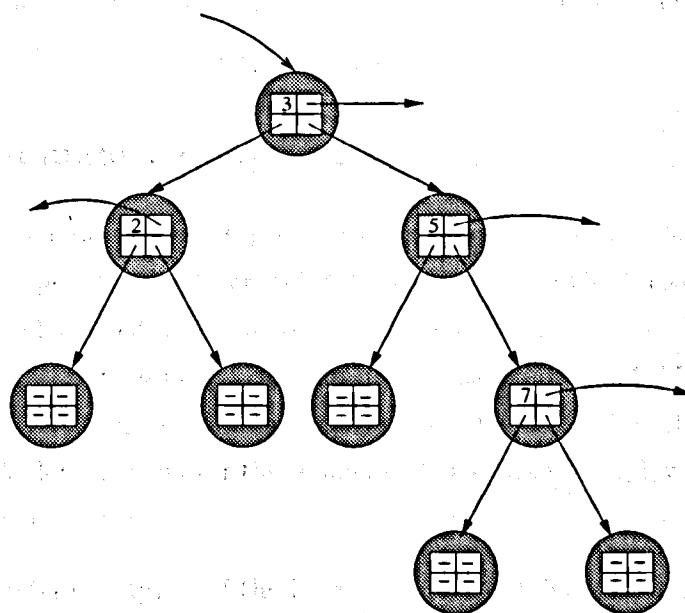


Figure 6.1: A symbol table

pair to be inserted, and the method **search** which returns the value associated with its key parameter (or **nil** if there is none).

When an object of this class is created all its variables have **nil** values and it assumes a quiescent state in which either of its methods may be invoked. When one object invokes a method in a second object, the activity of the first object is suspended until it is released from the rendezvous by execution of a **return** command by the second object or by some other object to which the responsibility for returning a result has been delegated. On completing the execution of a method body an object returns to its quiescent state; another method may then be invoked. Thus only one method may be active in an object at any point during computation. When an object α executes a **commit** command by invoking a method in an object β , it is implicit (i) that β should return its result not to α but to the object γ to which α should return a result, and (ii) that α is freed from the task of returning a result to γ . In particular, activity of α may proceed in parallel with that of β . Thus if the **search** method is invoked in a node with a key smaller (resp. larger) than that stored there, the node will commit that search to its left (resp. right) child, and we may think of the node as passing to the child the return address to which the result of the search should be sent. This address will have been received by the node either directly from the initiator of the search, if the node is the root, or from its parent in the tree. Since execution of a **return** or a **commit** command is not

the last action in the execution of a method body, objects may enjoy concurrent activity.

6.2 The semantic definition

We now give a semantics for the programming language by translation to the π_v -calculus. The type system of the calculus plays an important rôle in organizing the definition and reasoning about agents representing programs. We assume the existence of the infinite sets C containing all class names and M containing all method names. Further, we assume that there exists a partition of C into a finite set of blocks. We let κ range over these blocks. The types we employ in the semantic definition are given below.

1. The non-reference types of the language, i.e. `bool`, `int` and `unit`.
2. The sorts

| | |
|--------------|---|
| M_A^m | for $A \in C$ and $m \in M$, invocation links, |
| C_A | for $A \in C$, creation links, |
| R_T^κ | return links, |
| L_T | internal links, |
| E | external links, |

where T ranges over the types of the language. It is convenient to introduce the following synonyms: we write $NIL = L_{\text{unit}}$, $B = L_{\text{bool}}$ and $N = L_{\text{int}}$. The use of sorts will be explained below.

3. The record type $\{m_1 \vdash M_A^{m_1}, \dots, m_n \vdash M_A^{m_n}\}$, where m_1, \dots, m_n are the method names of class A . For notational simplicity we abbreviate this π_v -calculus record type to $\text{ref}(A)$. We refer to terms of sort $\text{ref}(A)$ as *object names* of class A . Such terms will be the process-calculus representations of ‘object identifiers’. If α is an object name of class A and $m \in \{m_1, \dots, m_n\}$ then the field selector $\alpha * m$ is of sort M_A^m . It is the link via which method m may be invoked in object α .

The values of the types are as follows: of the non-reference types the values `true`, `false`, `0`, `1`, `...` and `nil`; of the sorts, the names; and of the record types, the terms $\{\tilde{\ell} := \tilde{v}\}$ where \tilde{v} are names of sort M_A^m .

The sorting λ is the following partial function on sorts. First

$$\lambda(C_A) = \{(\text{ref}(A))\}$$

$$\lambda(R_T^\kappa) = \{(T)\}$$

$$\lambda(L_T) = \{(T)\}$$

$$\lambda(E) = \{(\text{int})\}$$

and if in the definition of class A we have method $m(Y_1:T_1, \dots, Y_n:T_n):T \dots$, and $A \in \kappa$, then

$$\lambda(M_A^m) = \{(T_1, \dots, T_n, R_T^\kappa)\}.$$

The significance of the sorting will become clear when the semantic mapping is given. Briefly, it is based on the following ideas:

1. A name of sort C_A may be used to communicate object names of class A . Such names will be used in the creation of objects.
2. A name of sort R_T^κ will represent a link via which a result of type T may be returned to an object of some class $A \in \kappa$.
3. A name of sort L_T may be used to communicate names of type T . Such names will be used for reading from and writing to the store.
4. A name of sort E may be used to communicate integers. Such names will be used in representing the interaction of a program with its environment.
5. A name of sort M_A^m will represent a link via which method m may be invoked in an object of class A . In an invocation a number of parameters are sent, represented by values of types T_1, \dots, T_n , and a link sort R_T^κ , along which the result should be returned.

We now define the translation $[\cdot]$ by induction on the structure of phrases. We will use L_T^n to abbreviate the n -tuple (L_T, \dots, L_T) . We begin with variable declarations. We have

$$[\text{var } X : T] : \text{abs}(L_T^3) \stackrel{\text{def}}{=} (r \ d \ w) \text{REG}_T\langle r, d, w, \text{nil} \rangle$$

where a register of type T is defined by

$$\begin{aligned} \text{REG}_T : \text{abs}(L_T^3, T) &\stackrel{\text{def}}{=} (r \ d \ w \ x) \\ &(\bar{r}\langle x \rangle. \text{REG}_T\langle r, d, w, x \rangle + \bar{d}\langle x \rangle. \text{REG}_T\langle r, d, w, \text{nil} \rangle \\ &+ w(y). \text{REG}_T\langle r, d, w, y \rangle). \end{aligned}$$

Here x represents the value stored, which is undefined, that is set to nil , when the variable is declared. The names r , d and w are links via which values of type T

may be communicated and they are used for reading, destructively reading, and writing the register respectively. A sequence of variable declarations is encoded as the composition of the agents representing the individual declarations.

An expression E of type T is encoded as an agent $\llbracket E \rrbracket : \text{abs}(\theta, L_T)$, where θ consists of sorts corresponding to the variables and new expressions that occur in E , whereas L_T corresponds to the name via which the value of the expression is to be delivered. For constants and variables of type T we have

$$\begin{aligned} \llbracket \text{nil} \rrbracket : \text{abs}(L_T) &\stackrel{\text{def}}{=} (val) \overline{val} \langle \text{nil} \rangle \\ \llbracket X \rrbracket : \text{abs}(L_T^2) &\stackrel{\text{def}}{=} (r \ val) r(x). \overline{val} \langle x \rangle \\ \llbracket X^! \rrbracket : \text{abs}(L_T^2) &\stackrel{\text{def}}{=} (d \ val) d(x). \overline{val} \langle x \rangle. \end{aligned}$$

Thus evaluation of a variable X involves reading the value of the variable by communication with the register corresponding to X via name r and delivering this value via name val . For the creation of a new object of class A ,

$$\llbracket \text{new}(A) \rrbracket : \text{abs}(C_A, L_A) \stackrel{\text{def}}{=} (n \ val) n(a). \overline{val} \langle a \rangle.$$

This agent receives via a name of sort C_A a term representing an object identifier of a new instance of class A and yields that term as its value. The intention is that the term will be received from the agent representing class A . Definitions of classes soon follow.

Next we look at operator definitions. Suppose $E_i : T_i$ and assume that $\llbracket E_i \rrbracket : \text{abs}(\theta_i, L_{T_i})$. Then for a unique θ with $\tilde{p} : \theta$, there are $\tilde{p}_i : \theta_i$ determined by the structure of the expressions so that

$$\begin{aligned} \llbracket \text{op}(E_1 \dots E_n) \rrbracket : \text{abs}(\theta, L_T) &\stackrel{\text{def}}{=} (\tilde{p} \ val) (\nu \ \tilde{w} : \text{NIL}, val_1 : L_{T_1}, \dots, val_n : L_{T_n}) \\ &\quad \llbracket E_1 \rrbracket \langle \tilde{p}_1, val_1 \rangle \\ &\quad | w_2. \llbracket E_2 \rrbracket \langle \tilde{p}_2, val_2 \rangle \\ &\quad | \dots \\ &\quad | w_n. \llbracket E_n \rrbracket \langle \tilde{p}_n, val_n \rangle \\ &\quad | val_1(x_1). \overline{w_2}. val_2(x_2) \dots \overline{w_n}. val_n(x_n). \overline{val} \langle \text{op}(\tilde{x}) \rangle. \end{aligned}$$

This definition captures the left to right evaluation of the operator's arguments as the last component receives the value of an expression E_i via an internal channel val_i and then activates evaluation of E_{i+1} by signalling on the internal link w_{i+1} . Once all expressions are evaluated, the operator is applied to their values and the result is returned via channel val . Similarly, method invocations may be defined:

$$\llbracket E_0!m(E_1 \dots E_n) \rrbracket : \text{abs}(\theta, L_T) \stackrel{\text{def}}{=} (\tilde{p} \ val) (\nu \ \tilde{w} : \text{NIL}, val_1 : L_{T_1}, \dots, val_n : L_{T_n})$$

$$\begin{aligned}
& [E_0] \langle \widetilde{p}_0, val_0 \rangle \\
& | w_1. [E_1] \langle \widetilde{p}_1, val_1 \rangle \\
& | \dots \\
& | w_n. [E_n] \langle \widetilde{p}_n, val_n \rangle \\
& | val_0(x_0). \overline{w_1}. val_1(x_1) \dots \overline{w_n}. val_n(x_n). \\
& (\nu r : R_T^\kappa) \overline{x_0} * \overline{m} \langle \widetilde{x}, r \rangle. r(v). \overline{val} \langle v \rangle
\end{aligned}$$

where $E_i : T_i$ and if $E_0 : \text{ref}(A)$, $E_0!m(\widetilde{E}) : T'$ then $A \in \kappa$, $T = T'$.

Each command S is encoded as an agent $[S] : \text{abs}(\theta, \text{NIL})$ where θ consists of sorts corresponding to the distinct variables and new expressions occurring in S as well as names associated with the return of results of method calls, and the final parameter (d in the definitions) is used to signal termination of S . We have the following:

$$\begin{aligned}
[X := E] : \text{abs}(\theta, L_T, \text{NIL}) & \stackrel{\text{def}}{=} (\widetilde{p} \ w \ d)(\nu \ val : L_T) \\
& [E] \langle \widetilde{p}, val \rangle \\
& | val(x). \overline{w} \langle x \rangle. \overline{d}
\end{aligned}$$

Hence, execution of an assignment consists of evaluation of the expression and communication with the store agent in order to write its value to the appropriate register. Similarly, the translation of a return command involving an expression of sort T and assuming that the penultimate parameter is the return link via which the result is to be returned, is

$$\begin{aligned}
[\text{return } E] : \text{abs}(\theta, R_T^\kappa, \text{NIL}) & \stackrel{\text{def}}{=} (\widetilde{p} \ r \ d)(\nu \ val : L_T) \\
& [E] \langle \widetilde{p}, val \rangle \\
& | val(x). \overline{r} \langle x \rangle. \overline{d}.
\end{aligned}$$

The translations of the remaining commands follow along similar lines, and in the commit statement we assume that $E_0 : \text{ref}(A)$ and $A \in \kappa$, $\widetilde{E} : \widetilde{T}$, and $E_0!m(\widetilde{E}_0) : T$.

$$\begin{aligned}
[\text{if } E \text{ then } S_1 \text{ else } S_2] : \text{abs}(\theta, \text{NIL}) & \stackrel{\text{def}}{=} (\widetilde{p}_0 \ \widetilde{p}_1 \ \widetilde{p}_2 \ d)(\nu \ val : B, d_1 : \text{NIL}, d_2 : \text{NIL}) \\
& [E] \langle \widetilde{p}_0, val \rangle \\
& | val(b). \text{cond}(b \triangleright \overline{d_1}. 0, \text{true} \triangleright \overline{d_2}. 0) \\
& | d_1. [S_1] \langle \widetilde{p}_1, d \rangle, \\
& | d_2. [S_2] \langle \widetilde{p}_2, d \rangle) \\
[S_1; S_2] : \text{abs}(\theta, \text{NIL}) & \stackrel{\text{def}}{=} (\widetilde{p}_1 \ \widetilde{p}_2 \ d)(\nu \ d' : \text{NIL})
\end{aligned}$$

$$\begin{aligned}
& [S_1] \langle \widetilde{p}_1, d' \rangle \\
& | d'. [S_2] \langle \widetilde{p}_2, d \rangle \\
\\
& [\text{commit } E_0!m(\widetilde{E})] : \text{abs}(\theta, R_T^K, \text{NIL}) \stackrel{\text{def}}{=} (\widetilde{p} \ r \ d)(\nu \widetilde{w} : \text{NIL}, \text{val}_1 : \mathbb{L}_{T_1}, \dots, \text{val}_n : \mathbb{L}_{T_n}) \\
& \quad [E_0] \langle \widetilde{p}_0, \text{val}_0 \rangle \\
& \quad | w_1. [E_1] \langle \widetilde{p}_1, \text{val}_1 \rangle \\
& \quad | \dots \\
& \quad | w_n. [E_n] \langle \widetilde{p}_n, \text{val}_n \rangle \\
& \quad | \text{val}_0(x_0). \overline{w_1}. \text{val}_1(x_1) \dots \overline{w_n}. \text{val}_n(x_n). \\
& \quad \quad \quad \overline{x_0 * m} \langle \widetilde{x}, r \rangle. \overline{d} \\
\\
& [\text{output } E] : \text{abs}(\theta, E, \text{NIL}) \stackrel{\text{def}}{=} (\widetilde{p} \ \text{out} \ d)(\nu \text{val} : \mathbb{N}) \\
& \quad [E] \langle \widetilde{p}, \text{val} \rangle \\
& \quad | \text{val}(v). \overline{\text{out}} \langle v \rangle, \overline{d} \\
\\
& [\text{input}] : \text{abs}(E, \text{NIL}) \stackrel{\text{def}}{=} (\text{in } \text{val}) \text{in}(x). \overline{\text{val}} \langle x \rangle
\end{aligned}$$

Note that in the representation of a sequential composition $S; S'$ the agent encoding of S interacts with the encoding of S' via a name of sort NIL to signal that S finishes and S' may begin.

It remains now to give the translations of class and program declarations. Suppose

$$Cdec ::= \text{class } B, Vdec, Mdec_s$$

where

$$Vdec ::= \text{var } X_1 : T_1, \dots, X_p : T_p,$$

$$Mdec_s ::= Mdec_1, \dots, Mdec_q,$$

and

$$Mdec_i ::= \text{method } m_i(\widetilde{Y}_i : \widetilde{T}_i) : T_i, \text{var } \widetilde{Z}_i : \widetilde{U}_i, S_i.$$

Then the encoding of a class declaration is as follows:

$$[Cdec] \stackrel{\text{def}}{=} (n_B \ \widetilde{n} \ \widetilde{z})! (\nu h_1 \dots h_q) \overline{n_B} \langle \alpha \rangle. \text{Obj}_B \langle \alpha, \widetilde{n}, \widetilde{z} \rangle$$

where $\alpha = \{m_1 := h_1, \dots, m_q := h_q\}$. The replicator $[Cdec]$ may repeatedly emit on the link n_B a new object name α of sort $\text{ref}(B)$ and thereby activate a new instance

of the class in its quiescent state, $\text{Obj}_B\langle\alpha, \tilde{n}, \tilde{z}\rangle$. The names \tilde{n} represent the creation links via which objects of the remaining classes may be created as required by $Cdec$, and $\tilde{z} = \langle\text{in}, \text{out}\rangle$ if $B = IO$ and $\langle\rangle$, otherwise. Agent Obj_B is defined as below.

$$\text{Obj}_B \stackrel{\text{def}}{=} (\alpha \tilde{n} \tilde{z})(\nu \tilde{p})([\text{var } X_1 : T_1, \dots, X_p : T_p][\tilde{p}] \mid \text{Body}_B\langle\alpha, \tilde{n}, \tilde{z}, \tilde{p}\rangle)$$

When a method m_i is invoked via the name $\alpha * m_i$, the appropriate method body is executed and an additional store containing the parameters of the method is created as follows.

$$\begin{aligned} \text{Body}_B \stackrel{\text{def}}{=} & (\alpha \tilde{n} \tilde{z} \tilde{p})(\nu d) \\ & (\Sigma_{i=1}^q \alpha * m_i(\tilde{v}_i, r).(\nu \tilde{u})) \\ & ((\Pi \text{REG}_{T_{ij}}\langle r_{Y_{ij}}, d_{Y_{ij}}, w_{Y_{ij}}, v_{ij}\rangle \mid [\text{var } \tilde{Z}_i : \tilde{U}_i])\langle \tilde{u} \rangle \mid [S_i]\langle \tilde{n}, \tilde{p}, \tilde{u}, m_i, r, d \rangle) \\ & \mid d. \text{Body}_B\langle\alpha, \tilde{n}, \tilde{p}\rangle) \end{aligned}$$

where $\tilde{u} = \tilde{r}_Y, \tilde{d}_Y, \tilde{w}_Y, \tilde{r}_Z, \tilde{d}_Z, \tilde{w}_Z$. The result of the method is returned via the name r supplied in the invocation and the completion of the execution of the method body is signalled via name $d : \text{NIL}$. The companion of this action is provided by the last component of the agent and results in the object returning to its quiescent state, while information concerning the method's local variables is lost, since the scope of the names \tilde{u} does not extend to the Body_B component. Finally we have the encoding of a program. If

$$Pdec \equiv Cdec_1, \dots, Cdec_r, \text{trigger } E$$

then

$$[Pdec] \stackrel{\text{def}}{=} (\tilde{n})([Cdec_1]\langle n_1, \tilde{n}_1 \rangle \mid \dots \mid [Cdec_r]\langle n_r, \tilde{n}_r \rangle \mid [IO]\langle \text{in}, \text{out} \rangle \mid [E]\langle \tilde{n}' \rangle)$$

where $\text{fn}([Pdec]) = \{\text{in}, \text{out}\}$, and \tilde{n}' represent the creation links via which objects may be created, as required by E .

6.3 Determinacy

We wish to isolate syntactic conditions on programs which guarantee determinacy. The behaviour of an object, being sequential, is determinate. Nondeterminism may arise if two objects α and β share a reference to a third object γ . For consider a state in a computation at which both α and β wish to invoke methods in γ . The subsequent behaviour of γ , and hence of the system, will in general not be independent of which invocation proceeds. We may expect, however, that if in no

state reachable from the initial configuration of a program do two objects share a reference, then the behaviour of the program is determinate. In fact we will be able to show that our conditions guarantee confluence.

We first state and explain the syntactic conditions.

Definition 6.3.1 Classes A_1, \dots, A_n form a *community* if each method body S of each A_j satisfies the following:

1. S does not contain $X := Y$ where X, Y are of type $\text{ref}(A_i)$,
2. S does not contain $E_0!m(\tilde{E}, X, \tilde{E}')$ or $\text{return } X$ where X is of type $\text{ref}(A_i)$,
3. if E contains $X!m(E_1, \dots, E_n)$ where $X : \text{ref}(A_i)$, then for no $h \in [1..n]$ does E_h have a subexpression $F_0!m'(\tilde{F}, X^\dagger, \tilde{F}')$, and
4. S is *responsible* (see below).

A program P_{dec} is a D -program if the classes declared in it form a community.

The first three conditions are concerned with the sharing of references. Condition 1 prevents a reference from being copied by an object. Note that an assignment of the form $X := Y^\dagger$ may appear in a D -program. Communication of a reference from one object to another can take place in two ways: as an argument of a method call or as a result of a method invocation. Condition 2 ensures that if an object sends a reference then it relinquishes it. Note that a D -program may contain statements similar to those prohibited by Condition 2 but which differ in that X^\dagger appears in the place of X . Moreover, in a statement of the form $X!m(E_1, \dots, E_n)$, where a method is to be invoked in the object to which the value of X is a reference, evaluation of E_1, \dots, E_n should not result in the reference to X being communicated to another object: this is the purpose of Condition 3. The purpose of Condition 4 is to prevent competition for access to an object through irresponsible activity related to the return of a method call. That is, if a method is invoked in an object, it should either return a result to the caller or delegate the responsibility for doing so to another object: it should not attempt to return a result more than once, nor should it attempt to return a result and delegate the responsibility for doing so to another object, nor should it delegate the responsibility to two or more objects. A command is *return/commit free*, *rcf*, if it contains no *return* command and no *commit* command.

Definition 6.3.2 The set of *responsible* commands is given as follows:

1. $\text{return } E$ and $\text{commit } E!m(\tilde{E})$ are responsible;

2. $S_1; S_2$ is responsible if exactly one of S_1, S_2 is responsible and the other is *rcf*;
3. if E then S_1 else S_2 is responsible if S_1, S_2 are responsible.

6.4 A D -program is confluent

The main result of this section is:

Theorem 6.4.1 Let $Pdec$ be a D -program. Then $\llbracket Pdec \rrbracket$ is fully db-confluent.

To prove the theorem it is necessary to undertake a detailed analysis of the behaviour of D -programs as defined by the translational semantics.

The following four lemmas capture a number of properties satisfied by the encodings of program fragments and will be useful in later reasoning. First, let us introduce some abbreviations. Let C be the union of all sorts C_A , L the union of all sorts L_T and given a block κ let R^* be the union of all sorts R_T^* . Further, we use σ to denote a partial function from variables to values, a store, and given $\sigma = \{(X_i, v_i)\}$, we define

$$\llbracket \sigma \rrbracket \langle \tilde{r}_i, \tilde{d}_i, \tilde{w}_i \rangle \stackrel{\text{def}}{=} \Pi \text{REG} \langle r_i, d_i, w_i, v_i \rangle.$$

It is also convenient to employ the following: Let $\alpha = \{m_1 := h_1 \dots m_q := h_q\} : \text{ref}(A)$, for some $A \in C$. We say that (i) $\alpha \in \text{fn}(P)$ if, for some j , $h_j \in \text{fn}(P)$ and (ii) α is borne (handled, controlled) in P if, for some j , h_j is borne (handled, controlled) in P . Moreover, we often write $(\nu\alpha)$ for $(\nu h_1 \dots h_q)$.

Lemma 6.4.2

1. Let E be an expression and $\llbracket E \rrbracket \equiv \llbracket E \rrbracket \langle \dots, val \rangle$. Suppose that for some σ , $P_0 = (\nu \tilde{z})(\llbracket E \rrbracket \mid \llbracket \sigma \rrbracket) \xRightarrow{s} P$, where P_0 is L -closed. Then $P \equiv (\nu \tilde{z})(Q \mid \llbracket \sigma' \rrbracket)$ and if val occurs in s then
 - (a) for all X , if X^\dagger occurs in E then $\sigma'X = \text{nil}$ otherwise $\sigma'X = \sigma X$, and
 - (b) $s = s_0 \overline{val} \langle v \rangle$ for some v , s_0 does not contain val , and $Q \equiv 0$.
2. Let S be a command and $\llbracket S \rrbracket \equiv \llbracket S \rrbracket \langle \dots, d \rangle$. Suppose that for some σ , $P_0 = (\nu \tilde{z})(\llbracket S \rrbracket \mid \llbracket \sigma \rrbracket) \xRightarrow{s} P$, where P_0 is L -closed. Then $P \equiv (\nu \tilde{z})(Q \mid \llbracket \sigma' \rrbracket)$ and if d occurs in s then $s = s_0 \overline{d}$, for some s_0 not containing d , and $Q \equiv 0$.

PROOF: The proofs are by structural induction on E and S and follow easily from the semantic definitions. \square

Recall that an object name of a class B is a term of the record type $\text{ref}(B)$ of the form $\{m_1 := h_1, \dots, m_n := h_n\}$ where m_1, \dots, m_n are the names of the methods of

B and $h_i : M_B^{m_i}$. It is convenient to fix, for each B in G, \tilde{A} , a partition \sim of the names of the M_B^m -sorts into blocks of the form $\{h_1, \dots, h_n\}$ where $h_i : M_B^{m_i}$, and to require that the names comprising an object name of type $\text{ref}(B)$ be those forming a block. Then we define a partition Π of Act by requiring that $\alpha \Pi \beta$ if $\alpha = \beta$, or $\alpha = r\langle v \rangle$ and $\beta = r\langle v' \rangle$ for some R-name r and some v, v' , or $\alpha = h\langle \tilde{v}, r \rangle$ and $\beta = h'\langle \tilde{v}', r' \rangle$ for some $h \sim h'$ and some $\tilde{v}, \tilde{v}', r, r'$. Thus two actions are equivalent if they represent receipt of results of a method invocation or invocations of methods in some object.

Lemma 6.4.3 Let B be a class. The following hold:

1. $\text{Obj}_B\langle \alpha \dots \rangle$ is fully convergent, consistent and Π -confluent.
2. For all derivatives P of $\text{Obj}_B\langle \alpha \dots \rangle$, P bears α and if P bears $\beta : \text{ref}(B)$ for some B , then $\alpha = \beta$. Further, if α occurs unguarded in subject position in P then P is of the form $(\nu \tilde{z})(\text{Body}_B(\tilde{u}) \mid [\sigma])$.
3. Let P be a derivative of $\text{Obj}_B\langle \alpha \dots \rangle$. If P handles $x : M_A^m$ then a subcommand $E!m(\tilde{E})$ where E is of type $\text{ref}(A)$ occurs in one of the method bodies of B .

PROOF: The first part follows from the fact that an object and therefore its encoding is a sequential entity. The only significant point is the guarded summation in its definition. This represents the willingness of the object, in its quiescent state, to accept any method invocation. The use of Π -confluence ensures that it is not required that the confluence property holds for different method invocations in an object. The second part follows easily from the translational definitions. Note that α is in fact the name of the object. The last part can be seen to hold from the translation and making use of the sorting. \square

Moreover, if P is a derivative of Obj_B and is of the form $(\nu \tilde{z})(\text{Body}_B(\tilde{u}) \mid [\sigma])$ we say that P is *quiescent*. Thus, in the quiescent state an object is capable of accepting method invocations.

Lemma 6.4.4 Let $[B]$ be the encoding of some class B . Then

$$[B] \stackrel{\text{def}}{=} (n_B \tilde{n} \tilde{z})!(\nu \alpha) \overline{n_B}(\alpha).D_j$$

where $\text{fn}(D_j) = \{\alpha, \tilde{n}, \tilde{z}\}$. Further, $[B]$ is a consistent, Π -confluent, L-closed agent.

PROOF: The result follows directly from the translational definitions and the previous lemma. \square

Lemma 6.4.5 Let $Pdec$ be a program declaration. Then $\llbracket Pdec \rrbracket$ is a friendly agent.

PROOF: Recall that if

$$Pdec \equiv Cdec_1, \dots, Cdec_r, \text{ trigger } E$$

then

$$\llbracket Pdec \rrbracket \stackrel{\text{def}}{=} (\nu \tilde{n})(\llbracket Cdec_1 \rrbracket \langle n_1, \tilde{n}_1 \rangle \mid \dots \mid \llbracket Cdec_r \rrbracket \langle n_r, \tilde{n}_r \rangle \mid \llbracket IO \rrbracket \langle \text{in}, \text{out} \rangle \mid \llbracket E \rrbracket \langle \tilde{n}' \rangle)$$

where $\text{fn}(\llbracket Pdec \rrbracket) = \{\text{in}, \text{out}\}$, and by the previous lemma, each of $\llbracket Cdec_i \rrbracket$ is a consistent replicator. Further, a derivative of $\llbracket Pdec \rrbracket$ has the form:

$$(\nu \tilde{q})(P_1 \mid \dots \mid P_n \mid \llbracket Cdec_1 \rrbracket \langle \dots \rangle \mid \dots \mid \llbracket Cdec_r \rrbracket \langle \dots \rangle \mid P)$$

where each of P_1, \dots, P_n is a derivative of some $\llbracket Cdec_j \rrbracket$ and P of $\llbracket E \rrbracket$. Thus by definition $\llbracket Pdec \rrbracket$ is a friendly agent. \square

As noted earlier, the confluence of a D -program relies on the absence of sharing of references to objects. We aim to prove that the conditions imposed on a community guarantee this fact and thereby establish the confluence of the encoding of a D -program. The first step towards this result considers evaluation of an expression occurring within a class of a community. This result has an assume-guarantee form: it asserts that assuming computation begins in an ‘acceptable’ state and satisfies certain properties, then it will not result in object-references being copied (Clause 1), or the sending of a reference that is retained within the body or the store (Clause 2(a)). Moreover, it will not send two copies of a reference (Clause 2(c)) and if it evaluates to a value of type $\text{ref}(A)$ then this value does not occur in the resulting store (Clause 3) and originates either from within the initial store or from an input action of the computation (Clause 4). By an acceptable state we mean that the store does not contain multiple copies of object references (Clause b). The property required of the environment (Clause c) is that it only provides inputs satisfying the following: if an input to state P carries an A -reference x then x is not already known to P . Hence the result ensures that evaluation of an expression will not be the first to violate conditions that guarantee absence of sharing. This result will later allow us to compose agents preserving this type of property and conclude their well-behavedness.

Lemma 6.4.6 Suppose classes $\tilde{A} = A_1, \dots, A_n$ form a community and E occurs in one of their method bodies. Suppose $\llbracket E \rrbracket \equiv \llbracket E \rrbracket \langle \dots, \text{val} \rangle$, $Q \equiv (\nu \tilde{z})(\llbracket E \rrbracket \mid \llbracket \sigma \rrbracket)$

and

$$Q \xrightarrow{\alpha_1} (\nu \tilde{z})(P_1 \mid [\sigma_1]) \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_m} (\nu \tilde{z})(P_m \mid [\sigma_m]) \xrightarrow{\overline{val}\langle v \rangle} (\nu \tilde{z})(P \mid [\sigma'])$$

where

- a. Q is L-closed,
- b. $\sigma \upharpoonright \text{ref}(\tilde{A})$ is injective, and
- c. if $\alpha_i = a\langle \tilde{x} \rangle$, $x_j : \text{ref}(A)$ then $x_j \notin \text{fn}((\nu \tilde{z})(P_{i-1} \mid [\sigma_{i-1}]))$, where $P_0 = [E]$ and $\sigma_0 = \sigma$.

Then for $1 \leq i \leq m$

1. $\sigma_i \upharpoonright \text{ref}(\tilde{A})$ is injective;
2. if $\alpha_i = (\nu \tilde{y})\bar{a}\langle \tilde{x}, r \rangle$ then
 - (a) if $x_j : \text{ref}(\tilde{A})$ then $x_j \notin \text{fn}((\nu \tilde{z})(P_i \mid [\sigma_i]))$,
 - (b) $r \in \tilde{y}$ and it is not handled by $(\nu \tilde{z})(P_i \mid [\sigma_i])$, and
 - (c) if $x_j, x_k : \text{ref}(\tilde{A})$ and $x_j = x_k$ then $j = k$;
3. if $E \neq X : \text{ref}(\tilde{A})$ and $v : \text{ref}(\tilde{A})$ then $v \notin \text{fn}([\sigma'])$;
4. if $v : \text{ref}(\tilde{A})$ then $v \in \text{fn}([\sigma]) \cup \{\tilde{x} \mid \text{for some } 1 \leq j \leq m, \text{ and some } a, \alpha_i = a\langle \tilde{x} \rangle\}$.

PROOF: The proof is by induction on the structure of E and Clauses 2 and 3 in the definition of a community play a crucial rôle.

In the cases $E \equiv \text{new}(B)$, X , X^\dagger , the result follows directly from the definitions.

If $E \equiv \text{op}(\tilde{E})$ the result follows easily by induction.

Next we consider the case $E \equiv E_0!m(E_1 \dots E_n)$, where $E_0 \neq X : \text{ref}(A)$. By definition $Q = (\nu \tilde{z})([E] \mid [\sigma])\langle \text{val} \rangle$ is given as follows:

$$\begin{aligned} Q \equiv & (\nu \tilde{z} \widetilde{val} \tilde{w})([E_0]\langle \text{val}_0 \rangle \\ & \mid w_1.[E_1]\langle \text{val}_1 \rangle \\ & \mid \dots \\ & \mid w_n.[E_n]\langle \text{val}_n \rangle \\ & \mid \text{val}_0(y_0). \overline{w_1}. \text{val}_1(y_1) \dots \overline{w_n}. \text{val}_n(y_n). \\ & (\nu r) \overline{y_0} * \overline{m}\langle \tilde{y}, r \rangle. r(u). \overline{val}\langle u \rangle \mid [\sigma]) \end{aligned}$$

Consider a computation of Q satisfying assumption (c).

$$\begin{aligned}
Q &\xrightarrow{s_0} Q_0 \equiv (\nu \tilde{z} \widetilde{val} \tilde{w})([E_1] \langle val_1 \rangle \\
&\quad | \dots \\
&\quad | w_n. [E_n] \langle val_n \rangle \\
&\quad | val_1(y_1) \dots \overline{w_n}. val_n(y_n). \\
&\quad (\nu r) \overline{x_0 * m} \langle \tilde{y}, r \rangle. r(u). \overline{val} \langle u \rangle \mid [\sigma_1]) \\
&\xrightarrow{s_1} Q_1 \equiv (\nu \tilde{z} \widetilde{val} \tilde{w})([E_2] \langle val_2 \rangle \\
&\quad | \dots \\
&\quad | w_n. [E_n] \langle val_n \rangle \\
&\quad | val_2(y_2) \dots \overline{w_n}. val_n(y_n). \\
&\quad (\nu r) \overline{x_0 * m} \langle x_1 \tilde{y}, r \rangle. r(u). \overline{val} \langle u \rangle \mid [\sigma_2]) \\
&\xrightarrow{s_2} \dots \\
&\xrightarrow{s_n} Q_n \equiv (\nu \tilde{z} \widetilde{val} \tilde{w})((\nu \tilde{y} r) \overline{x_0 * m} \langle \tilde{x}, r \rangle. r(u). \overline{val} \langle u \rangle \mid [\sigma_{n+1}]) \\
&(\nu \tilde{y} r) \overline{x_0 * m} \langle \tilde{x}, r \rangle Q_{n+1} \equiv (\nu \tilde{z}') (r(u). \overline{val} \langle u \rangle \mid [\sigma_{n+1}]) \\
&\xrightarrow{r(v)} Q_{n+2} \equiv (\nu \tilde{z}') (\overline{val} \langle v \rangle \mid [\sigma_{n+1}]) \xrightarrow{\overline{val} \langle v \rangle} Q' \equiv (\nu \tilde{z}') (0 \mid [\sigma_{n+1}])
\end{aligned}$$

where

$$(\nu \tilde{z})([E_i] \mid [\sigma_i]) \xrightarrow{s_i} \xrightarrow{\overline{val}_i \langle x_i \rangle} (\nu \tilde{z})(0 \mid [\sigma_{i+1}])$$

and $\sigma_0 = \sigma$. By the induction hypothesis, for all i , if $x_i : \text{ref}(\tilde{A})$ then

$$x_i \in \text{fn}([\sigma_i]) \cup \{\tilde{x} \mid \text{for some } \beta \in s_i, \beta = b\langle \tilde{x} \rangle\}$$

and $\sigma_i[\text{ref}(\tilde{A})]$ is injective. In addition, by Clause 2 in the definition of a community, $E_i \neq X : \text{ref}(\tilde{A})$, for $1 \leq i \leq n$. Hence by Property 3 of the induction hypothesis, if $x_i : \text{ref}(\tilde{A})$ then $x_i \notin \text{fn}([\sigma_{i+1}])$, and so by Lemma 6.4.2(1),

* if $x_i : \text{ref}(\tilde{A})$ then $x_i \notin \text{fn}([\sigma_j])$ for all $j > i$.

First consider the transition $Q_{n+1} \xrightarrow{r(v)} Q_{n+2}$. By condition (c) of the lemma, $v \notin \text{fn}(Q_{n+1})$ and so $v \notin \text{fn}(Q')$. Therefore Properties 3 and 4 are satisfied.

So let R be a derivative of Q such that, for some $i < n$ and $s'_i \leq s_i$,

$$Q_i \xrightarrow{s'_i} R = (\nu \tilde{z} \widetilde{val} \tilde{w})(P \mid \dots \mid val_i(x_i) \dots (\nu r) \overline{x_0 * m} \langle \tilde{x}, r \rangle. r(u). \overline{val} \langle u \rangle \mid [\sigma_{ij}])$$

where

$$(\nu \tilde{z})([E_i] \mid [\sigma_i]) \xrightarrow{s'_i} (\nu \tilde{z})(P \mid [\sigma_{ij}]).$$

By induction, $\sigma_{ij}[\text{ref}(\tilde{A})]$ is injective. So Property 1 holds. To prove Property 2 suppose additionally that

$$R \xrightarrow{\alpha} R' = (\nu \tilde{z} \widetilde{val} \tilde{w})(P' \mid \dots \mid val_i(x_i) \dots (\nu r) \overline{x_0 * m} \langle \tilde{x}, r \rangle. r(u). \overline{val} \langle u \rangle \mid [\sigma_{ij}])$$

where $\alpha = (\nu \tilde{p})\bar{a}\langle \tilde{y}, r' \rangle$. By the induction hypothesis and Property 2 we conclude that

- i. if $y_k : \text{ref}(\tilde{A})$ then $y_k \notin \text{fn}((\nu \tilde{z})(P' \mid [\sigma_{ij}]))$,
- ii. r' is not handled by $(\nu \tilde{z})(P \mid [\sigma_{ij}])$, and
- iii. if $y_k, y_l : \text{ref}(\tilde{A})$ and $y_k = y_l$ then $k = l$.

Moreover, it is easy to see that

- iv. if $y_k : \text{ref}(\tilde{A})$ then $y_k \in \text{fn}([\sigma_i]) \cup \{\tilde{x} \mid \text{for some } \beta \in s'_i, \beta = b\langle \tilde{x} \rangle\}$.

First, we want to show that if $y_i : \text{ref}(\tilde{A})$ then $y_i \notin \text{fn}(R') = \text{fn}((\nu \tilde{z})(P' \mid [\sigma_{ij}]) \cup \{x_0, \dots, x_{i-1}\})$. By observation (*) above, if $x_m : \text{ref}(\tilde{A})$ then $x_m \notin \text{fn}([\sigma_i])$ for all $m \leq i - 1$. Furthermore, since $x_m \in \text{fn}(T)$ for all T such that $Q_i \xRightarrow{s} T \xrightarrow{\tau} R$, by assumption (c), $x_m \notin \{\tilde{x} \mid \text{for some } \beta \in s'_i, \beta = b\langle \tilde{x} \rangle\}$. Thus by properties (i) and (iv), 2(a) is satisfied. A combination of (ii) and the fact that r is a new name not handled in R guarantees Property 2(b) and Property 2(c) follows straightforwardly from (iii).

So finally, we need to consider transition $Q_n \xrightarrow{(\nu \tilde{y}r)\bar{x}_0 \dots \bar{x}_m(\tilde{x}, r)} Q_{n+1}$. By observation (*), Property 2(a) is satisfied and, clearly, Property 2(b) also holds. On the other hand, if $x_i, x_j : \text{ref}(\tilde{A})$ where $i < j$ then by observation (*), $x_i \notin \text{fn}([\sigma_j])$. By assumption (c) and since $x_i \in \text{fn}(Q_j)$, $x_i \notin \{\tilde{x} \mid \text{for some } \beta \in s_j, \beta = b\langle \tilde{x} \rangle\}$. Since $x_j \in \text{fn}([\sigma_j]) \cup \{\tilde{x} \mid \text{for some } \beta \in s_j, \beta = b\langle \tilde{x} \rangle\}$, $x_i \neq x_j$ as required. This establishes Property 2(c) and completes the case.

Finally we consider the case $E \equiv X!m(\tilde{E})$, $X : \text{ref}(\tilde{A})$. The proof of this is similar to that of the previous case with one difference: considering the execution of $Q = (\nu \tilde{z})([E] \mid [\sigma])\langle \text{val} \rangle$, which is similar to the one considered above, the value of X , $x_0 \in \text{fn}([\sigma_0])$. So in this case, in order to establish Property 2(a), we need to prove that, in the syntax of the execution of the previous case, if $Q \xRightarrow{s} Q_j \xrightarrow{\alpha \langle \dots x_0 \dots \rangle} Q'_j$ then $j = n + 1$ and $\alpha = x_0$. Suppose $E_i \neq X^\dagger$, for all $i < j$, for some j . Then by Clause 3 in the definition of a community, X^\dagger does not occur in E_i for all $i < j$ and, by Lemma 6.4.2 we deduce that $x_0 = \sigma X = \sigma_1 X = \dots = \sigma_j X$. Hence, Property 2(a) guarantees that x_0 has not been sent out during the transition $Q \xRightarrow{s} Q_j$. Alternatively, if $E_j = X^\dagger$ then $\sigma_{j+1} X = \text{nil}$. This implies that x_0 will not occur in any transition during the evaluation of the remainder of the parameters of the method invocation, as it actually occurs free in the derivatives of this computation. This allows us to complete the proof using arguments as in the previous case. \square

A similar result may be proved for the execution of a command occurring in a community.

Lemma 6.4.7. Suppose classes $\tilde{A} = A_1, \dots, A_n$ form a community and S occurs in one of their method bodies. Suppose $[S] \equiv [S](\bar{p}, d)$, $Q \stackrel{\text{def}}{=} (\nu \tilde{z})([S] \mid [\sigma])$ and

$$Q \xrightarrow{\alpha_1} (\nu \tilde{z})(P_1 \mid [\sigma_1]) \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_m} (\nu \tilde{z})(P_m \mid [\sigma_m]) \xrightarrow{\bar{d}} (\nu \tilde{z})(P \mid [\sigma'])$$

where

- a. Q is L-closed;
- b. $\sigma \upharpoonright \text{ref}(\tilde{A})$ is injective;
- c. if $\alpha_i = a\langle \tilde{x} \rangle$, $x_j : \text{ref}(A_j)$ then $x_j \notin \text{fn}((\nu \tilde{z})(P_{i-1} \mid [\sigma_{i-1}]))$, where $P_0 \equiv [S]$, $\sigma_0 = \sigma$.

Then for all $1 \leq i \leq m$

- 1. $\sigma_i \upharpoonright \text{ref}(\tilde{A})$ is injective;
- 2. (a) if $\alpha_i = (\nu \tilde{y})\bar{a}\langle \tilde{x}, r \rangle$ and $x_j : \text{ref}(\tilde{A})$ then $x_j \notin \text{fn}((\nu \tilde{z})(P_i \mid [\sigma_i]))$, and if $x_j = x_k$ then $j = k$;
- (b) if $\alpha_i = \bar{r}\langle x \rangle$, $x : \text{ref}(\tilde{A})$ then $x \notin \text{fn}((\nu \tilde{z})(P_i \mid [\sigma_i]))$.

PROOF: This is by structural induction on S and uses Clauses 1 and 2 in the definition of a community.

If $S \equiv \text{return } E, X := E$ then by Clauses 1 and 2 in the definition of a community, $E \neq Y : \text{ref}(\tilde{A})$ and the claim follows easily using the previous lemma. We consider the case $S \equiv \text{return } E$. By definition $Q \equiv (\nu \tilde{z})([S] \mid [\sigma])(r, d)$ is given by:

$$Q = (\nu \tilde{z}d')([E]\langle d' \rangle \mid d'(v). \bar{r}\langle v \rangle. \bar{d} \mid [\sigma])$$

A computation of Q satisfying assumption (c) looks as follows:

$$Q \xRightarrow{s} Q_1 = (\nu \tilde{z})(\bar{r}\langle x \rangle. \bar{d} \mid [\sigma']) \xrightarrow{\bar{r}\langle x \rangle} Q_2 = (\nu \tilde{z})(\bar{d} \mid [\sigma']) \xrightarrow{\bar{d}} Q_3 = (\nu \tilde{z})[\sigma']$$

where

$$(\nu \tilde{z})([E]\langle d' \rangle \mid [\sigma]) \xRightarrow{s} \xrightarrow{\bar{d}'\langle x \rangle} (\nu \tilde{z})[\sigma'].$$

By Lemma 6.4.6(1), $\sigma' \upharpoonright \text{ref}(\tilde{A})$ is injective. Property 2(a) follows by Lemma 6.4.6(2). Finally, Property 2(b) follows from Lemma 6.4.6(3) and the fact that $E \neq Y : \text{ref}(\tilde{A})$ which is imposed by Clause 2 in the definition of a community.

If $S \equiv E_0!m(\tilde{E})$, $\text{commit } E_0!m(\tilde{E})$ then the proof appeals to the translational definitions and uses the same arguments as the previous lemma.

If $S \equiv \text{if } E \text{ then } S_1 \text{ else } S_2$ or $S \equiv S_1; S_2$ then the proof follows easily by induction. \square

We may now extend this result to the case of the encoding of an object. The following lemma asserts that when an object is placed in a well-behaved environment, if it emits an object reference then it relinquishes it, and it never sends out two copies of the same reference. The condition imposed on the environment is that it will not provide the object with two copies of a reference in a transition representing a method invocation nor with object references the object already knows. Note that in this case no conditions are imposed on the state of the initial store of the object: by definition all variables are initially set to nil.

Lemma 6.4.8 Suppose classes $\tilde{A} = A_1, \dots, A_n$ form a community and $P = \text{Obj}_B\langle\alpha\rangle$ where $B \in \{A_1, \dots, A_n\}$. Suppose $P \xrightarrow{\alpha_1} P_1 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_m} P_m$ where if $\alpha_i = a\langle\tilde{x}\rangle$ then

- a. if $x_j : \text{ref}(\tilde{A})$ then $x_j \notin \text{fn}(P_i)$;
- b. if $x_j, x_k : \text{ref}(\tilde{A})$ and $x_j = x_k$ then $j = k$.

Then for all $1 \leq i \leq m$

- 1. if $\alpha_i = (\nu \tilde{y})\bar{a}\langle\tilde{x}, r\rangle$ and $x_j : \text{ref}(\tilde{A})$, then x_j is not controlled in P_i , and if $x_j = x_k$ then $j = k$;
- 2. if $\alpha_i = \bar{r}\langle x\rangle$, $x : \text{ref}(\tilde{A})$, then x is not controlled in P_i .

PROOF: The proof follows easily from the translational definition and Lemma 6.4.7. \square

Having dealt with object links it remains to show that return links do not come to be shared as computation proceeds. This fact solely relies on the *responsibility* of commands as we will see in the following analysis.

Lemma 6.4.9

- 1. Suppose S is a responsible command and let P be a derivative of $[S]$. If $P \xrightarrow{\alpha} P'$ then
 - (a) if $\alpha = \bar{r}\langle x\rangle$ or $\alpha = (\nu \tilde{y})\bar{a}\langle\tilde{x}, r\rangle$, where $r \notin \tilde{y}$, then $r \notin \text{fn}(P')$;
 - (b) if $\alpha = (\nu \tilde{y}r)\bar{a}\langle\tilde{x}, r\rangle$ then r appears only in positive subject position in P' .

2. Suppose A is a class each of whose method bodies is responsible and let P be a derivative of Obj_A . If $P \xrightarrow{\alpha} P'$ then

- (a) if $\alpha = a\langle\tilde{x}, r\rangle$ then r is not borne in P' ;
- (b) if $\alpha = (\nu\tilde{y})\bar{r}\langle x\rangle$ or $\alpha = (\nu\tilde{y})\bar{a}\langle\tilde{x}, r\rangle$, where $r \notin \tilde{y}$, then $r \notin \text{fn}(P')$;
- (c) if $\alpha = (\nu\tilde{y}r)\bar{a}\langle\tilde{x}, r\rangle$, then r appears only in positive subject position in P' .

PROOF: The proof of the first property is by structural induction on S and it makes use of the following facts which are implied by the semantic definition:

- i. For all expressions E , $\llbracket E \rrbracket$ does not handle any names of sort R^κ for any κ ;
- ii. If S is an *rcf* command then $\llbracket S \rrbracket$ does not handle any names of sort R^κ for any κ ;
- iii. If P is a derivative of $\llbracket E!m(\tilde{E}) \rrbracket$ and $P \xrightarrow{\alpha} P'$, $\alpha = (\nu\tilde{y})\bar{a}\langle\tilde{x}, r\rangle$, then $r \in \tilde{y}$ and it appears only in positive subject position in P' .

The following possibilities exist:

- $S \equiv \text{return } E$. By definition,

$$\llbracket S \rrbracket = (r d)(\nu d')(\llbracket E \rrbracket\langle d' \rangle \mid d(v). \bar{r}\langle v \rangle. \bar{d}. 0).$$

It is easy to see by construction and (i) that Property 1(a) is satisfied. Property 1(b) follows from (iii).

- $S \equiv \text{commit } E!m(\tilde{E})$. By definition,

$$\begin{aligned} \llbracket S \rrbracket = & (r d)(\nu \tilde{d} \tilde{v})(\llbracket E \rrbracket\langle v_0 \rangle \\ & \mid d_1. \llbracket E_1 \rrbracket\langle v_1 \rangle \\ & \dots \\ & \mid d_n. \llbracket E_n \rrbracket\langle v_n \rangle \\ & \mid v_0(x_0). \bar{d}_1 \dots \bar{d}_n. v_n(x_n). \overline{x_0 * m}\langle \tilde{x}, r \rangle. \bar{d}. 0). \end{aligned}$$

The result follows using argument similar to the previous case.

- $S \equiv S_1; S_2$. By definition,

$$\llbracket S \rrbracket = (r d)(\nu d')(\llbracket S_1 \rrbracket\langle d' \rangle \mid d' \llbracket S_2 \rrbracket\langle d \rangle)$$

By the definition of responsible, at least one of S_1 and S_2 is *rcf*. If S_1 is *rcf* then by induction observation (ii), Property 1 holds. Similarly, if S_2 is *rcf* then by (ii), $r \notin \text{fn}(\llbracket S_2 \rrbracket)$ and so Property 1 holds.

- $S \equiv \text{if } E \text{ then } S_1 \text{ else } S_2$. This case follows similarly by induction.

This completes the proof of Property 1.

The proof of Property 2 follows as a corollary of this fact and the semantic definition. Note in particular that when a method is invoked in a quiescent object then the return link supplied in the invocation will either be passed to another object due to the effect of a commit command, or it will be used to return the result of the invocation. It will not be used in a positive subject position (Clause 2(a)). \square

We now formulate the invariants satisfied by derivatives of the encoding of a D -program which guarantee that sharing does not occur.

Theorem 6.4.10 Suppose $Pdec$ is a D -program. Then for every derivative S of $\llbracket Pdec \rrbracket$, $S \equiv (\nu \tilde{p})(\Pi P_i \mid \Pi !C_j \mid J)$ and the following hold:

1. $\text{fn}(S) = \{\text{in}, \text{out}\}$, in is borne by J and out is handled by J , and neither occurs in any other component of S ,
2. J and each P_i is fully convergent, Π -confluent and a derivative of some $!C_j$,
3. if $a : C$ then a is uniquely handled in S , and
4. if $a : \text{ref}(\tilde{A})$ for some A or $a : R^\kappa$ then a is persistently managed in S .

PROOF: We sketch the proof of the result. Suppose

$$Pdec \equiv Cdec_1, \dots, Cdec_r, \text{ trigger } E_0$$

where $E_0 = \text{new}(A)!m(\text{new}(IO))$. Then the encoding of $Pdec$ is given as follows:

$$\llbracket Pdec \rrbracket \equiv (\nu \tilde{p})(\llbracket Cdec_1 \rrbracket(\tilde{p}_1) \mid \dots \mid \llbracket Cdec_r \rrbracket(\tilde{p}_r) \mid \llbracket E_0 \rrbracket(\dots))$$

and $\llbracket E_0 \rrbracket \stackrel{\text{def}}{=} n_A \langle x \rangle. n_{IO} \langle y \rangle. (\nu r) \overline{x} * \overline{m} \langle y, r \rangle. r \langle v \rangle. 0$.

Property 1 is a result of our assumption that the commands input and output(E) may only occur within the IO class of which there exists exactly one object during computation. Property 2 is guaranteed by Lemma 6.4.3 and the translational definition. Property 3 is a consequence of the sorting: since $C \notin \tilde{i} \in \lambda(I)$ for all sorts I , no names of sort C are passed in communications during the computation of a program and since all such names are uniquely handled in the initial state, this property is preserved. Finally, Property 4 follows by induction on the length of the derivation and a combination of Proposition 2.5.8 with Lemma 6.4.8, for the result concerning $\text{ref}(\tilde{A})$ -names and Lemma 6.4.9, for the result concerning R^κ -names. Note that the clause of the proposition concerning sorts is satisfied as by 1, if $x \in \text{fn}(S)$, for S a derivative of $\llbracket Pdec \rrbracket$, then $x : E$ and $\lambda(E) = (\text{int})$. \square

Proof of Theorem 6.4.1

According to the previous results, the encoding of a D -program satisfies the clauses of Definition 2.5.10. Hence it is effective. In addition, by Theorem 6.4.10(2), it also satisfies the conditions of Theorem 3.8.5 hence $[D]$ is Π -confluent. Since $[D]$ is M, R -closed, by Lemma 3.8.3(3), the claim follows. \square

6.5 Transformations

Consider the following class declaration:

```
class T'
var K:int, V:ref(A), L:ref(T'), R:ref(T')
method insert(X:int, W:ref(A)):unit
  if K=nil then (K:=X ; V:=W ; L:=new(T') ; R:=new(T') )
  else if X=K then V:=W
        else if X<K then L!insert(X,W)
        else R!insert(X,W) ;
  return nil
method search(X:int):ref(A)
  if K=nil then return nil
  else if X=K then return V
        else if X<K then return L!search(X)
        else return R!search(X)
```

which is a variant of the following class (repeated from Section 6.1)

```
class T
var K:int, V:ref(A), L:ref(T), R:ref(T)
method insert(X:int, W:ref(A)):unit
  return nil ;
  if K=nil then (K:=X ; V:=W ; L:=new(T) ; R:=new(T) )
  else if X=K then V:=W
        else if X<K then L!insert(X,W)
        else R!insert(X,W)
method search(X:int):ref(A)
  if K=nil then return nil
  else if X=K then return V
        else if X<K then commit L!search(X)
        else commit R!search(X)
```

We may view the second class as being obtained from the first by the application of three transformations: in the `insert` method a command of the form $S; \text{return } E$ is transformed to $\text{return } E; S$, and in the `search` method two commands of the form $\text{return } E!m(\tilde{E})$ are transformed to $\text{commit } E!m(\tilde{E})$.

In contrast to class T , class T' is such that in both of its methods the `return` and `commit` statements are the last actions of the method body. As a result class T' prescribes systems of a sequential nature. For consider a tree composed of objects of class T' and suppose method `insert` is invoked in the root of the tree. This will result in a chain of invocations being initiated down a branch of the tree where each node, except the last, is waiting for a reply from its child within the chain before it responds to its own parent. Only when the insertion takes place in the appropriate tree node will the replies trickle from child to parent to eventually release the root and subsequently the original caller from the rendezvous. Thus we may see that as an effect of the invocation the entire tree becomes blocked and no other object may interact with the data structure until the insertion is completed. On the other hand, when an `insert` method is invoked in the root of a tree composed of T -nodes, the root releases the caller from the rendezvous *before* proceeding with the insertion. Thus within a tree of objects of this second class, many insertions may take place concurrently. The effect of invoking the `search` method is similar, locking the data structure in the former case and allowing concurrent activity in the latter case.

The classes T and T' are taken from [Jon93a] with the significant change that, as in [LW95a], the values associated with the integer keys are references to objects of an arbitrary class A as opposed to integers, as is the case in [Jon93a]. In the same paper, Jones gives a formal development of class T' from a specification and then derives T by applying two program transformation rules. The correctness of the transformation was then pursued in [LW95a], where the following result was proved:

Theorem 6.5.1 Let P_{dec} be an arbitrary program of the language in which the first class T above is declared. Let P_{dec}' be the program obtained from it by replacing the declaration of that class by that of the second class T above. Then $\llbracket P_{dec} \rrbracket \simeq \llbracket P_{dec}' \rrbracket$.

This body of work has raised the challenging question under what conditions this type of transformation rules may be applied and how the existing techniques for reasoning about programs may be extended and used to prove their soundness. Thus, our aim is to enunciate syntactic conditions under which transformations of

the forms

$$S; \text{return } E \rightsquigarrow \text{return } E; S \quad \text{and} \quad \text{return } E!m(\tilde{E}) \rightsquigarrow \text{commit } E!m(\tilde{E})$$

may be safely applied, and to prove that this is the case. To begin to do this we first note that we must take account of the possibility of non-termination of method invocations. For consider the following program declaration *Pdec*:

```
class A
method m(V:IO):unit
  new(A)!m(V) ; return nil ; V!Out(3)

trigger new(A)!m(new(IO))
```

Pdec prescribes a single non-terminating computation in which nothing is output to the environment. If, however, the first transformation is applied to the body of method *m*, so that it becomes `return nil ; new(A)!m(V) ; V!Out(3)`, the resulting program has computations in which output is produced. Moreover, if instead in *Pdec* the body of method *m* is changed to `return new(A)!m(V) ; V!Out(3)` to obtain *Pdec'* and the second transformation is then applied so that the body becomes `commit new(A)!m(V) ; V!Out(3)`, the resulting program again has computations in which output is produced although nothing is output in the single non-terminating computation of *Pdec'*.

The criterion of correctness we will adopt will in fact consider the transformations *not* to alter the observable behaviours of *Pdec* and *Pdec'*. The reason for this is that both they and their respective transformed variants prescribe *divergent* systems which may proceed indefinitely without interacting with the environment. Thus rather than using branching bisimilarity as the criterion of indistinguishability of behaviour we will use its divergence-sensitive variant, introduced in Chapter 2.

To move towards the syntactic conditions we first examine why the transformations cannot be applied arbitrarily. Consider the following program declaration *Pdec*:

```
class A
var X:ref(A), Y:ref(B)
method m(V:IO):unit
  X:=new(A) ; Y:=new(B) ; Y!init() ; return X!inca(Y) ;
  V!Out(Y!read())
method inca(Z:ref(B)):unit
```

```
Z!incb() ; return nil
```

```
class B
```

```
var W:int
```

```
method init():unit
```

```
  W:=0 ; return nil
```

```
method incb():unit
```

```
  W:=W+1 ; return nil
```

```
method read():int
```

```
  return W
```

```
trigger new(A)!m(new(I0))
```

The single computation of $Pdec$ results in the trigger creating an object α of class A which creates objects β of class A and γ of class B, and 1 being output to the environment. Suppose $Pdec'$ is obtained from it by applying the first transformation to the body of method *inca* of class A, resulting in `return nil ; Z!incb()`. The output on executing $Pdec'$ could be either 1 or 0, the latter if α invokes *read* in γ before β invokes *incb* in γ , something which is not possible in $Pdec$ as in that case β must invoke *incb* in γ before freeing α to invoke *read* in γ . Further, if $Pdec''$ is obtained from $Pdec$ by applying the second transformation to the body of method *m* of class A, resulting in `return X!inca(Y)` being replaced by `commit X!inca(Y)`, then again the output could be either 1 or 0 as by committing to β the responsibility for returning a result to the trigger, α frees itself to invoke method *read* in γ before β invokes *incb*.

These simple examples strongly suggest that syntactic conditions sufficient to guarantee safety of the transformations should prohibit sharing of references to some extent. In Section 6.3, in the definition of a community, we identified syntactic conditions that prohibit the sharing of object references and showed that they guarantee confluence of programs conforming to them. We would like to insert a community within a wider program context while preserving the property that references to objects of that community are not shared. However, while objects of a community may not be directly responsible for the creation of shared references to objects of that community, other objects may be. The following definition strengthens the notion of 'community' to prevent this latter possibility.

Definition 6.5.2 Let $Pdec$ be a program declaration whose classes are G, \tilde{A}, \tilde{N} . In $Pdec$ the classes G, \tilde{A} form a guarded community with guard G if:

1. the classes G, \tilde{A} form a community;
2. no method body of a class in \tilde{N} contains $\text{new}(A_i)$;
3. no method of class G has a parameter or result type $\text{ref}(A_i)$;
4. if $E!m(\tilde{E})$ occurs in a method body of the classes G, \tilde{A} , then E is of type $\text{ref}(A_i)$;
5. there exists a block κ of the partition of C , such that $G, \tilde{A} \subseteq \kappa$ and $\tilde{N} \cap \kappa = \emptyset$.

The intention is that within a system prescribed by such a program, references to \tilde{A} -objects cannot be shared although other references may be. Condition 1 ensures that objects within the guarded community are not directly responsible for the creation of sharing of \tilde{A} -references. Conditions 2–4 prevent \tilde{A} -references from being acquired by \tilde{N} -objects (which might otherwise act in such ways as to share them): \tilde{N} -objects are unable to create references to \tilde{A} -objects (condition 2); \tilde{N} -objects can interact with \tilde{A} -objects only by invoking methods in G -objects, and G -objects do not return \tilde{A} -references (condition 3); no G -object or \tilde{A} -object can invoke a method in a G -object or an \tilde{N} -object (condition 4), necessary as G -objects and \tilde{A} -objects can share references to \tilde{N} -objects, and references to G -objects can be shared. Note that as a result, no G -object or \tilde{A} -object can interact with the program's environment as they are unable to invoke methods in the IO -object. The last condition is not necessary for the correctness of the transformations. Its purpose is to facilitate the proof of correctness by allowing us to distinguish the return links present in the encoding of a guarded community: they are of sort R_T^* . According to the translational definition, such names may be borne but not controlled in the encoding of the classes \tilde{N} , as we will see in detail later.

The subsystem generated by a guarded community within a program is a well-behaved agent (an important fragment of it is Π -confluent). One might wonder if the transformations can always be applied safely within classes comprising a guarded community. In fact this is not the case as can be seen by considering the following program declaration *Pdec*:

```
class N
method m(V:IO):unit
  new(G)!m0() ; return nil ; V!Out(3)

class G
var X:ref(A)
```



```

method m0():unit
  X:=new(A) ; X!m1(X†) ; return nil

```

```

class A
method m1(Y:ref(A)):unit
  new(A)!m2(Y†) ; return nil
method m2(Y:ref(A)):unit
  return Y!m3()
method m3():unit
  return nil

```

```

trigger new(N)!m(new(IO))

```

Execution results in the trigger creating an object α of class N which creates an object β of class G which creates an object γ of class A which creates an object δ of class A . The system deadlocks without producing output: δ can only invoke method $m3$ in γ , but γ can only receive the result of its invocation of method $m2$ in δ . However, if the first transformation is applied to the body of method $m1$, yielding `return nil ; new(A)!m2(Y†)`, the resulting program outputs 3. If instead $Pdec'$ is obtained from $Pdec$ by replacing the body of $m1$ with `return new(A)!m2(Y†)`, again the program deadlocks without output, while if the second transformation is applied to the body of $m1$ of $Pdec'$, yielding `commit new(A)!m2(Y†)`, again 3 is output. This motivates the following condition.

Definition 6.5.3 Let $Pdec$ be a program declaration in which the classes G, \tilde{A} form a guarded community. G, \tilde{A} form a society if no method body in G, \tilde{A} contains an expression $X!m(\tilde{E}, X^\dagger, \tilde{E}')$.

It is in fact the case, though it is not obvious, that a system generated by a society (which for convenience we refer to also as a 'society') cannot have 'cycles' of references or return links of a certain kind and hence that a method invocation in a society cannot fail because of deadlock. This will be explained further in a later section. Finally we can state the transformation rules.

Transformation 6.5.4 Suppose G, \tilde{A} form a society in a program $Pdec$.

1. A command of the form `return E!M(\tilde{E})` may be replaced by `commit E!M(\tilde{E})` in a method body of G, \tilde{A} .

2. A command of the form $S; \text{return } E$ may be replaced by $\text{return } E; S$ in a method body of G, \tilde{A} provided no variable X occurs (as X or X^\dagger) in both S and E .

We now turn to proving their correctness.

6.6 Correctness

We aim to establish the following correctness result.

Theorem 6.6.1 Let $Pdec$ be a program declaration in which the classes G, \tilde{A} form a society. Let $Pdec'$ be obtained from it by applying the transformations an arbitrary number of times to method bodies of G, \tilde{A} . Then $[Pdec] \simeq_1 [Pdec']$.

In order to prove this theorem it is sufficient to prove the correctness of a single application of the transformations. The proof consists of a careful and rigorous analysis of the systems in question and the theory of partial confluence plays a central part in it. We let M be the union of all sorts M_c^m , $c \in G, \tilde{A}$, and R the union of all sorts R_T^κ , where κ is the block of C such that $G, \tilde{A} \subseteq \kappa$.

So let $Pdec$ be a program declaration in which G, \tilde{A} form a society. Let $Pdec'$ be obtained from it by applying a single transformation to one of the method bodies of G, \tilde{A} . Then $[Pdec]$ is of the form $(\nu \tilde{p})(P \mid I)$ where $I = (\nu n_{\tilde{A}})[G, \tilde{A}]$ and with P encoding the remaining classes and the trigger, and $[Pdec']$ is $(\nu \tilde{p})(P \mid I')$ where I' is the encoding of the transformed society. Note that it is appropriate to restrict on the names $n_{\tilde{A}}$ in the encoding of the society as the context of the society is not able to create any \tilde{A} objects.

Let \mathcal{P} be the transition graph generated by P and \mathcal{I} the transition graph generated by I and I' . We will establish the following:

1. \mathcal{P} is R -polite.
2. \mathcal{P} is (M^-, R^+) -ready.
3. \mathcal{I}^R is (M^+, R^-) -disciplined.

The final hypothesis of Theorem 4.1.26, M, R -closure, follows easily from the sorting respected in the semantic definition and Lemma 2.5.7. Hence, by that theorem,

$$[Pdec] \simeq_1 (\nu \tilde{p})(P \mid I^b) \quad \text{and} \quad [Pdec'] \simeq_1 (\nu \tilde{p})(P \mid I'^b),$$

where I^b and I'^b are the states corresponding to I and I' in $\mathcal{I}^{\leq 1}$. Thus to prove $\llbracket Pdec \rrbracket \simeq_1 \llbracket Pdec' \rrbracket$ it suffices to show that the agents I^b and I'^b are indistinguishable in an arbitrary program context. This would be most directly achieved if we could show that $I^b \simeq_1 I'^b$: the result would follow as \simeq_1 is preserved by the operators. In fact this does hold if the agents are fully convergent, but not in general.

To understand why the result does not hold in general suppose the first transformation is applied to $S; \text{return } E$ where S is the only source of divergence (consider for instance a minor variation of the class A in the first example of Section 6.5). Then $I^b \uparrow$ but $I'^b \downarrow$ as the result will be returned via some action $\rho : R^-$ before the diverging computation begins. Note, however, that $I'^b \uparrow \rho$. Dually, if the second transformation is applied in a body of the form $\text{return } E!m(\tilde{E}); S$ where S is the only source of divergence, then this time $I^b \downarrow$ (but $I^b \uparrow \rho$) while $I'^b \uparrow$ as the replacement of return by commit unguards the divergence. Although $I^b \not\simeq_1 I'^b$ in such cases, the slight differences in divergent behaviour are lost in a program context which, being ready, can contribute R^+ -actions to turn any R^- -divergence of I^b into divergence of $(\nu \tilde{p})(P \mid I^b)$, and similarly for I'^b . So instead we can establish the following:

$$3. I^b \simeq_1^{R^-} I'^b.$$

Then by Theorem 4.2.3, the result follows.

It now remains to prove the three claims above.

6.6.1 A guarded community

The aim of this section is to establish properties satisfied by the encoding of a guarded community and it makes use of the results of Section 6.4, where the confluence of a D -program was established. The main distinction between a guarded community and a D -program is that the former may communicate with an environment capable of invoking methods in the community's objects. Moreover, it may be given access to objects that are outside the community. Thus we need to ensure that these interactions do not result in the sharing of a reference to a community object and indeed that the community will not attempt to initiate interaction with non-community objects, access to which may be shared. It turns out that a guarded community is not Π -confluent. This is because the environment may provide the community with the same return link for more than one method invocation. This violates the unique controlling property and may result in non-confluent behaviour. Nonetheless, we may show that a fragment of the labelled transition system corresponding to the encoding of a guarded community is Π -confluent. Specifically, if I is

the encoding of a guarded community then I^R is Π -confluent. This is sufficient as in the particular application we are considering, where the environment is a program context, the guarded community will be provided only with fresh return links.

In the next lemma we formalize some properties concerning the free names of the encoding of a guarded community. These may be names of objects of the guarding class G and return links via which results to methods invoked via these names are to be returned. Another free name of the agents in question is n_G via which object names of class G may be communicated representing the creation of G -objects. Other free names may also exist corresponding to object names of classes other than G and \tilde{A} . The lemma characterizes the way in which these occur. The proof makes use of the sorting and Condition 3 in the definition of a guarded community which implies that

* if $x : M_G^m$ and $\lambda(M_G^m) = \{(\tilde{I}, R_T^x)\}$ then $T \neq \text{ref}(A_i)$ and $I_j \neq \text{ref}(A_i)$ for all i, j .

Lemma 6.6.2 Suppose classes G, A_1, \dots, A_n form a guarded community. Then for every derivative $S^R = (\nu \tilde{p})(\Pi P_j \mid \llbracket G \rrbracket \mid \Pi \llbracket A_i \rrbracket)^R$ of $S_0^R = (\nu \tilde{n}_A)(\llbracket G \rrbracket \mid \Pi \llbracket A_i \rrbracket)^R$, if $x \in \text{fn}(S)$ one of the following holds:

1. $x : \text{ref}(G)$ and x is uniquely borne and not controlled in S , or
2. $x : C_G$ and x is uniquely controlled in S , or
3. $x : R_T^x$ where $T \neq \text{ref}(A_i)$ for all i , and x is uniquely controlled and not borne in S , or
4. $x : C_D$ where $D \notin G, \tilde{A}$ and x is not controlled in S , or
5. $x : \text{ref}(D)$ where $D \notin G, \tilde{A}$, and x does not occur in subject position in S .

Further, S_0 is $\text{ref}(\tilde{A})$ -closed.

PROOF: First we note that by Lemma 6.4.3(3) and Clause 4 in the definition of a guarded community, if x occurs in S and $x : \text{ref}(B)$, where $B \notin \tilde{A}$, then x is not controlled in S . Further, it is easy to see, by the sorting and Lemma 2.5.7, that S_0 is $\text{ref}(\tilde{A})$ -closed. We prove the remainder of the claim by induction on the length of $S_0 \xRightarrow{*} S$. Clearly, the claim holds for the base case. So suppose $S_0 \xRightarrow{*} S' = (\nu \tilde{p}')P' \xrightarrow{\alpha} S = (\nu \tilde{p})P$ for some $P, P', \tilde{p}, \tilde{p}'$. Several cases exist:

- if $\alpha = a\langle \tilde{x}, r \rangle$ then by the induction hypothesis, $a : \text{ref}(G)$. Hence, by observation (*) above, if $r : R_T^x$ then $T \neq \text{ref}(A_i)$ for all i , and if $x_i : \text{ref}(B)$ then

$B \notin \tilde{A}$. So x_i is not controlled in S . This establishes Property 5. Further, since S' is a derivative of S_0^R , $r \notin \text{fn}(S')$ and r is uniquely controlled and not borne in S . This implies Property 3. The remaining properties follow by induction.

- if $\alpha = \bar{r}\langle x \rangle$ then, by the induction hypothesis and Property 3, $r : R_T^\kappa$ and as a result $x : T$ where $T \notin \tilde{A}$. Since the free names of S' have not been affected by the induction hypothesis the claim holds.
- if $\alpha = (\nu\beta)\overline{n_G}\langle\beta\rangle$ then $\beta : \text{ref}(G)$ and the claim trivially holds.
- if $\alpha = n_D\langle\gamma\rangle$ where $n_D : C_D$ and $D \notin \tilde{A}$ then $\gamma : \text{ref}(D)$ where $D \notin \tilde{A}$. As observed earlier, γ is not controlled within S so Property 4 holds and the remaining properties follow by induction.
- if $\alpha = \tau$ then the claim follows by the induction hypothesis and Lemma 6.4.8 which guarantee that internal actions do not result in the violation of the unique controlling and unique bearing of free names. \square

We continue to take a closer look at the bound names of the encoding of a guarded community.

Lemma 6.6.3 Suppose classes G, A_1, \dots, A_n form a guarded community. Then for every derivative $S = (\nu\tilde{p})(\Pi P_j \mid \llbracket G, \tilde{A} \rrbracket)$ of $S_0 = (\nu\tilde{n}_A)(\llbracket G, \tilde{A} \rrbracket)$ the following hold:

1. for all j there exists $\text{Obj}\langle\alpha\rangle$ such that $\text{Obj}\langle\alpha\rangle \xrightarrow{\alpha_1} P_{j1} \dots \xrightarrow{\alpha_m} P_{jm} = P_j$ where if $\alpha_i = a\langle\tilde{x}\rangle$ and $x_k : \text{ref}(A)$, then $x_k \notin \text{fn}(P_{ji})$ and if $x_k = x_l$ then $k = l$;
2. if $x \in \text{bn}(S) \cap \text{fn}(\Pi P_j)$ and x occurs in subject position in S , then one of the following holds
 - (a) $x : R_T^\kappa$ and x is uniquely borne and uniquely controlled in S ;
 - (b) $x : \text{ref}(A_i)$ for some i , and x is uniquely borne and uniquely controlled in S ;
 - (c) $x : C_{A_i}$ and x is borne in S .
3. if $x \in \text{bn}(S) \cap \text{fn}(\llbracket G, \tilde{A} \rrbracket)$ then $x : C_{A_i}$ for some i , and x is uniquely controlled in S .

PROOF: By induction on the length of $S_0 \xrightarrow{s} S$. The base case is clearly true. So suppose S is a derivative of S_0 and $S = (\nu\tilde{p})(\Pi P_j \mid \llbracket G, \tilde{A} \rrbracket) \xrightarrow{\alpha} S'$. The following possibilities exist:

- $\alpha = a\langle\tilde{x}, r\rangle$ and $P_1 \xrightarrow{\alpha} P'_1$. By Lemma 6.6.2, S is $\text{ref}(\tilde{A})$ -closed, so, if $x_i : \text{ref}(B)$ then $B \notin \tilde{A}$ and Property 1 is satisfied. Since the bound names of the agent have not been affected, the result follows by the induction hypothesis.
- If $\alpha = \bar{r}\langle x\rangle, n_D\langle\gamma\rangle, (\nu\beta)\overline{n_G}\langle\beta\rangle$ the case follows easily as above.
- $\alpha = \tau$. The transition may be the result of an internal transition of a P_i or a communication between the components of S as follows:
 1. $S' = (\nu\tilde{p})(P'_1 \mid P_2 \mid \dots \mid P_n \mid [G, \tilde{A}])$, $P_1 \longrightarrow P'_1$. Clearly, the first part of the lemma holds. The remaining properties hold by induction due to the fact that the transition involves no communication of names between the components of S .
 2. $S' = (\nu\tilde{p})(P'_1 \mid P_{n+1} \mid P_2 \mid \dots \mid P_n \mid [G, \tilde{A}])$, $P_1 \xrightarrow{n(\beta)} P'_1, [G, \tilde{A}] \xrightarrow{(\nu\beta)\overline{n_G}\langle\beta\rangle} [G, \tilde{A}] \mid P_{n+1}$ where $P_{n+1} = (\nu\tilde{q})\text{Obj}\langle\beta\rangle$. Since β is a new name, it is uniquely borne by P_{n+1} and controlled by at most P_1 . Therefore, by induction the claim is satisfied.
 3. $S' = (\nu\tilde{p})(P'_1 \mid P'_2 \mid P_3 \mid \dots \mid P_n \mid [G, \tilde{A}])$, $P_1 \xrightarrow{\bar{r}\langle x\rangle} P'_1$ and $P_2 \xrightarrow{r\langle x\rangle} P'_2$. By Lemma 6.4.8(3), if $x : \text{ref}(A)$ then $x \notin \text{fn}(P'_1)$. Since by induction, $x \notin \text{fn}(P_i)$ for $i \neq 1$, x is uniquely controlled in S' by component P'_2 . By induction the remainder of the claim follows.
 4. $S' = (\nu\tilde{p})(P'_1 \mid P'_2 \mid P_3 \mid \dots \mid P_n \mid [G, \tilde{A}])$, $P_1 \xrightarrow{(\nu\tilde{y})\overline{\alpha}\langle\tilde{x}, r\rangle} P'_1$, $P_2 \xrightarrow{\alpha\langle\tilde{x}, r\rangle} P'_2$ and $r \notin \tilde{y}$. By Lemma 6.4.8(2), if $x_i : \text{ref}(A)$ then P'_1 does not handle x_i and if $x_k = x_i$ then $k = i$. Hence Property 1 is satisfied. Also by the induction hypothesis x_i is not controlled by P_j , $j \neq 1$. Thus, it is uniquely controlled in S' by P_2 . In addition, by the same lemma, P'_1 does not handle r . Hence, r is uniquely controlled by P'_2 in S' . The remainder of the lemma follows by induction.
 5. $S' = (\nu\tilde{p})(P'_1 \mid P'_2 \mid P_3 \mid \dots \mid P_n \mid [G, \tilde{A}])$, $P_1 \xrightarrow{(\nu\tilde{y}r)\overline{\alpha}\langle\tilde{x}, r\rangle} P'_1$ and $P_2 \xrightarrow{\alpha\langle\tilde{x}, r\rangle} P'_2$. This is similar to the previous case and makes use of Lemma 6.4.8(1).

□

To sum up, the encoding of a guarded community satisfies the following properties:

Theorem 6.6.4 Suppose classes G, A_1, \dots, A_n form a guarded community. Then for every derivative $S = (\nu\tilde{p})(\Pi P_j \mid [G, \tilde{A}])$ of S_0^R , $S_0 = (\nu\tilde{n}_A)([G, \tilde{A}])$, the following hold:

1. if $x \in \text{fn}(S)$ then x is uniquely controlled or uniquely borne in S but not both;

2. if $x \in \text{fn}(P_i) - \text{fn}(S)$ then x is uniquely controlled and uniquely borne in S ;
3. for all $B \in G, \tilde{A}$, $\llbracket B \rrbracket = !(\nu \alpha) \overline{n_j}(\alpha).D_j$, where $\text{fn}(D_j) - C = \{\alpha\}$ and n_j is uniquely handled in S ;
4. each P_i is convergent, fully Π -confluent and a derivative of $\llbracket B \rrbracket$ where $B \in G, \tilde{A}$.

□

Thus given the encoding of a guarded community I , we have that I^R is effective. By Theorem 3.9.5 we may conclude the following:

Corollary 6.6.5 I^R is fully Π -confluent. □

In particular this implies that for any guarded community with encoding I , I^R , the fragment of the state space of the community capable of handling at most one method invocation at any time, is in fact Π -confluent.

6.6.2 Deadlock freedom

As observed earlier a guarded community may be liable to deadlock caused by a certain type of cycles consisting of return and reference links. The precise definition of the cycles in question is given below:

Definition 6.6.6 Suppose G, \tilde{A} form a guarded community and let $S = (\nu \tilde{p})(\Pi P_i \mid \llbracket G, \tilde{A} \rrbracket)$ be a derivative of $(\nu n_{\tilde{A}})(\llbracket G, \tilde{A} \rrbracket)$. We define \ll as follows: $P_i \ll P_j$ if there exists x such that either $x : \text{ref}(\tilde{A})$, P_i controls x and P_j bears x , or $x : R$, P_i bears x and P_j controls x . We say that S has a cycle if there exist $P_{i_1} \dots P_{i_n}$ such that $P_{i_1} \ll P_{i_2} \ll \dots \ll P_{i_n} \ll P_{i_1}$.

We may prove that a society is cycle-free.

Lemma 6.6.7 Suppose G, \tilde{A} form a society. Then for every derivative $S = (\nu \tilde{p})(\Pi P_i \mid \llbracket G, \tilde{A} \rrbracket)$ of $S_0 = (\nu n_{\tilde{A}})(\llbracket G, \tilde{A} \rrbracket)$ the following hold:

1. if $P_i \xrightarrow{(\nu r) \overline{x * m}(\tilde{y}, r)} P'_i$ and $y_j : \text{ref}(\tilde{A})$ then $x \neq y_j$;
2. if P_i bears $\alpha : \text{ref}(\tilde{A})$ then α does not occur in P_i in object position;
3. S has no cycles.

PROOF: The proof of the first property follows similarly to that of Lemma 6.4.8 making additional use of the properties of a society.

We now consider the proof of the last two properties. This is by induction on the length of the derivation of $S_0 \xRightarrow{*} S$. The base case is clearly true. So suppose

$S_0 \xrightarrow{s} Q \xrightarrow{\alpha} S$ where $Q = (\nu \tilde{q})(\Pi P_i \mid [G, \tilde{A}])$ and Q satisfies the properties of the lemma. The following possibilities exist for the transition $Q \xrightarrow{\alpha} S$:

1. $S = (\nu \tilde{p})(P'_1 \mid P_2 \mid \dots \mid P_n \mid [G, \tilde{A}])$ where $P_1 \xrightarrow{\alpha} P'_1$. By Lemma 6.6.2 one of the following possibilities exist:
 - $\alpha = \tau, \bar{r}(x)$. Clearly, Property 2 is satisfied and since the transition has not resulted in the ownership of further names by any component, Property 3 is also satisfied.
 - $\alpha = n_D(b)$ where $D \neq A_i$ for all i . Then by the sorting $b : \text{ref}(D)$ and since the transition has not affected the R -names and $\text{ref}(\tilde{A})$ -names of the system, Properties 2 and 3 are satisfied.
 - $\alpha = a * m(\tilde{x}, r)$ where $a : \text{ref}(G)$. By Lemma 6.4.3(2), since P_1 bears $a : \text{ref}(G)$, it bears no names of sort $\text{ref}(\tilde{A})$. Hence, by induction, Property 2 is satisfied. To establish the third property we recall that according to the sorting, if $x_i : \text{ref}(B)$ then $B \notin \tilde{A}$. Hence, the ownership of names of sort $\text{ref}(\tilde{A})$ has not been affected within S . Moreover, by Lemma 6.6.2(3), r is not borne by any component of S . Thus Property 3 is satisfied.
2. $\alpha = (\nu \gamma) \bar{n}(\gamma)$ and $S = (\nu \tilde{p})(P_1 \mid P_2 \mid \dots \mid P_n \mid P_{n+1} \mid [G, \tilde{A}])$, where $[G] \xrightarrow{(\nu \tilde{\gamma}) \bar{n}(\gamma)} [G] \mid P_{n+1}$. It is easy to see that the properties are satisfied.
3. $\alpha = \tau$ and $S = (\nu \tilde{p} \beta)(P'_1 \mid P_2 \mid \dots \mid P_n \mid P_{n+1} \mid [G, \tilde{A}])$, where $P_1 \xrightarrow{n(\beta)} P'_1$ and $[G, \tilde{A}] \xrightarrow{(\nu \beta) \bar{n}(\beta)} [G, \tilde{A}] \mid P_{n+1}$. It is easy to see that the properties are satisfied.
4. $\alpha = \tau$ and $S = (\nu \tilde{p})(P'_1 \mid P'_2 \mid \dots \mid P_n \mid [G, \tilde{A}])$

$$P_1 \xrightarrow{(\nu \tilde{y} r) \bar{a} * m(\tilde{x}, r)} P'_1 \text{ and } P_2 \xrightarrow{a * m(\tilde{x}, r)} P'_2.$$

First note that by Lemma 6.6.3(2)(b), $a : \text{ref}(\tilde{A})$. By Property 1, $x_i \neq a$ for all i . Thus Property 2 is satisfied by S . We prove Property 3 by contradiction. Suppose there exists a cycle in S . There are two possible cases: either the cycle has been created due to the ownership of an $x_i : \text{ref}(\tilde{A})$ by P'_2 , or due to the ownership of r by P'_2 . Consider the former case. According to the definition of a cycle, there exist P_{i_1}, \dots, P_{i_k} such that $P'_2 \ll P_{i_1} \ll \dots \ll P_{i_k} \ll P'_2$, where P_{i_1} bears x_{i_1} . Consider P_{i_k} . By definition of \ll , either P_{i_k} controls a name x of sort $\text{ref}(\tilde{A})$, which is borne by P'_2 or, P_{i_k} bears a name $r : R$ which is controlled by P'_2 . By Lemma 6.4.3(2), P'_2 bears exactly one name of sort $\text{ref}(\tilde{A})$, namely a and since a occurs unguarded in subject position in P_2 , by the same lemma P_2 is quiescent and $\text{fn}(P_2) - C = \{a\}$. This implies that r is the only name

of sort R controlled by P'_2 . By Lemma 6.6.3(2)(b), a is uniquely controlled by P'_1 and r is uniquely borne by P'_1 . Thus, we conclude that $P_{i_k} = P'_1$. Since P_1 controls name x_i , $P_1 \ll P_{i1}$. In addition, since $P_{i1} \ll \dots \ll P'_1$, it is easy to see that $P_{i1} \ll \dots \ll P_1$. Hence, there exists a cycle within Q which is a contradiction.

For the latter case suppose that a cycle exists in S containing $P'_1 \ll P'_2$ due to the sharing of name r by the two components. Since P_1 controls a and P_2 bears a , a similar cycle containing $P_1 \ll P_2$ would exist in Q . This contradicts our induction hypothesis, hence no cycles exist in S .

5. $S = (\nu \tilde{p})(P'_1 \mid P'_2 \mid \dots \mid P_n \mid [G, \tilde{A}])$, where

$$P_1 \xrightarrow{(\nu \tilde{y}) \overline{a * m}(\tilde{x}, r)} P'_1 \text{ and } P_2 \xrightarrow{a * m(\tilde{x}, r)} P'_2$$

and $r \notin \tilde{y}$. Suppose for the sake of contradiction that a cycle has been created within S due the ownership of name r by P'_2 . Then there must exist P_{i_1}, \dots, P_{i_k} such that $P_{i_1} \ll \dots \ll P_{i_j} \ll P'_2 \ll \dots \ll P_{i_k} \ll P_{i_1}$, where P_{i_j} bears r . Since P_1 controls r and $a * m$ we have that $P_{i_1} \ll \dots \ll P_{i_j} \ll P_1 \ll P_2 \ll \dots \ll P_{i_k} \ll P_{i_1}$. Hence, a cycle exists within Q which is a contradiction. The rest of the proof follows as in the previous case.

6. $S = (\nu \tilde{p})(P'_1 \mid P'_2 \mid \dots \mid P_n \mid [G, \tilde{A}])$, where $P_1 \xrightarrow{\tilde{r}(x)} P'_1$ and $P_2 \xrightarrow{\tilde{r}(x)} P'_2$. The interesting case is when $x : \text{ref}(\tilde{A})$. First, we observe that if P'_2 bears $y : \text{ref}(B)$ then $x \neq y$, otherwise the cycle $P_1 \ll P_2 \ll P_1$ would have existed in Q . This guarantees that Property 2 holds. The proof that S has no cycles follows similarly to case 4. \square

We may also prove that a society is not prone to deadlock. We will show that for each derivative I of $I_0 = (\nu n_{\tilde{A}})[G, \tilde{A}]$, representing a state where exactly one method call originating outside of the society has not yet been returned, either I is divergent, because it has a non-terminating computation or because it has a computation in which a run-time error occurs, or it can return the result of the method invocation. Moreover, if the system is convergent after returning the result, it may reach a state in which each object-agent has become quiescent.

Lemma 6.6.8 Let I be a derivative of I_0 which contains a single free name r of sort R , where r is uniquely controlled in I , representing that exactly one method invocation originating outside the society has not yet been returned and that r is the name via which the return is to be made. Suppose that $I \downarrow$. Then $I \Rightarrow \xrightarrow{\tilde{r}(v)} I'$

for some I' and v . Moreover, if also $I \downarrow \bar{r}(v)$ then $I \xrightarrow{\bar{r}(v)} I^q$, where each object-agent component of I^q represents an object in its quiescent state.

PROOF: Consider a derivative $I = (\nu \tilde{p})(\Pi P_i \mid [G, \tilde{A}])$ of $(\nu n_{\tilde{A}})[G, \tilde{A}]$, such that I has exactly one free name r of sort R , where r is uniquely controlled in I . By Lemma 6.4.3 and Lemma 6.6.3 one of the following holds for I :

1. there exists i such that $P_i \xrightarrow{\bar{r}(x)} P'_i$, or
2. for each i , either P_i represents an object in its quiescent state, or P_i can engage in one of the following actions: τ , $n\langle\alpha\rangle$, $\overline{ret}\langle x\rangle$, $ret\langle x\rangle$, $\overline{a * m}\langle \tilde{x}, ret\rangle$, or $(\nu ret)\overline{a * m}\langle \tilde{x}, ret\rangle$, where $r \neq ret$.

In the former case we have that $I \xrightarrow{\bar{r}(x)}$ and the proof of the first part of the lemma is complete. In the latter case we claim that $I \xrightarrow{\tau} I'$, where $I' \downarrow$ and r is the only name of sort R that occurs free in I' . First we note that since I handles r , there exists at least one j such that P_j is not quiescent. Suppose for the sake of contradiction that $I \not\xrightarrow{\tau}$. Then it must be that for all i , if P_i is not quiescent then it can engage in an action of the three latter types (note that if a component were able to engage in one of the first three types of actions then a communication involving the action would take place giving $I \xrightarrow{\tau}$). So, there exist $i_1 \dots i_n$, such that for all j , $P_{i_j} \xrightarrow{\alpha}$, where $\alpha = \overline{a * m}\langle \tilde{x}, ret\rangle$, $(\nu ret)\overline{a * m}\langle \tilde{x}, ret\rangle$, or $ret\langle x\rangle$. Moreover, if $\alpha = \overline{a * m}\langle \tilde{x}, ret\rangle$ then a is uniquely borne by some P_{i_k} , where $P_{i_k} \xrightarrow{a * m\langle \tilde{x}, ret\rangle}$ and similarly if $\alpha = ret\langle x\rangle$ then ret is uniquely controlled by some P_{i_k} where, $P_{i_k} \xrightarrow{ret\langle x\rangle}$. It is then easy to see that the P_{i_1}, \dots, P_{i_n} contain a cycle within I , which contradicts the previous lemma. So $I \xrightarrow{\tau}$ as required. Since $I \downarrow$ there exists I' such that $I \Rightarrow I' \not\xrightarrow{\tau}$. Thus I' satisfies the first clause above and $I' \xrightarrow{\bar{r}(x)} Q$.

If in addition $I \downarrow r$ then $I \Rightarrow \xrightarrow{\bar{r}(x)} Q \downarrow$, where by construction, Q satisfies the second property above. Using a similar argument we deduce that $Q \Rightarrow Q'$, where each object-agent component of Q' represents an object in its quiescent state. This completes the proof. \square

6.6.3 Partial Confluence

Lemma 6.6.9 P is R -polite.

PROOF: A derivative T of P is a composition whose components are the replicator encodings of the class definitions not belonging to a society, derivatives of them representing objects in various states, and a derivative of the trigger process. The names of sort R are the return links for invocations via names of sort M_c^m where

$c \in G, \tilde{A}$. We look into the way these names are used and mentioned. We claim that if an R-name occurs free in T then it appears only in positive subject position and T is of the form

$$T \equiv (\nu \tilde{p})(r_1(x_1).P_1 \mid \dots \mid r_n(x_n).P_n \mid T')$$

where $r_1 \dots r_n : R$ are pairwise distinct. We prove this by induction on the length of the derivation.

The base case is clearly true. For the induction step suppose $P \xRightarrow{\bullet} S' \xrightarrow{\alpha} S$ where

$$S' \equiv (\nu \tilde{p})(r_1(x_1).P_1 \mid \dots \mid r_n(x_n).P_n \mid T').$$

There exist the following possibilities for the transition $S' \xrightarrow{\alpha} S$.

1. If $\alpha \notin M^- \cup R^+$ then $\alpha = \tau, n_G\langle\beta\rangle, \text{in}\langle x\rangle, \overline{\text{out}}\langle x\rangle, (\nu \tilde{y})\bar{c}\langle \tilde{x}, r\rangle, \bar{r}\langle y\rangle$, where $c : M_n^m$ and $r : R_T^\mu, \mu \neq \kappa$. It is direct to see that this transition does not affect the R-names of Q and clearly S is of the form required.
2. If $\alpha \in M^-$, and $\alpha = (\nu \tilde{y}r)\overline{a * m}\langle \tilde{x}, r\rangle$, then r may easily be seen to occur free and unguarded within Q' only in positive subject position:

$$S \equiv (\nu \tilde{p})(r(x).P \mid r_1(x_1).P_1 \mid \dots \mid r_n(x_n).P_n \mid T')$$

Note that $Q \not\xrightarrow{\alpha}$ where $\alpha = (\nu \tilde{y})\overline{a * m}\langle \tilde{x}, r\rangle, r \in \text{fn}(Q)$ since, by the induction hypothesis, for all $r : R \in \text{fn}(Q)$, r is not controlled in Q .

3. If $\alpha \in R^+$ then it must be that $\text{subj}(\alpha) = r_i$ for some $r_i \in r_1 \dots r_n$ and

$$S \equiv (\nu \tilde{p})(r_1(x_1).P_1 \mid \dots \mid P_i \mid \dots \mid r_n(x_n).P_n \mid T').$$

Thus, we may see from (2) above that, for all derivatives of P' of P , if $P' \xrightarrow{(\nu \tilde{y})\overline{a}\langle \tilde{x}\rangle}$ and $x_i : R$ then $x_i \in \tilde{y}$, and so P is R-polite. \square

Lemma 6.6.10 \mathcal{P} is (M^-, R^+) -ready

PROOF: As illustrated in the previous lemma, for all derivatives T of P , if an R-name occurs free in T then it appears only in positive subject position. Further, T has the form

$$T \equiv (\nu \tilde{p})(r_1(x_1).P_1 \mid \dots \mid r_n(x_n).P_n \mid T')$$

where $r_1 \dots r_n : R$ are pairwise distinct since they are generated as private names by actions of the form $(\nu \tilde{y}r)\overline{m}\langle \tilde{x}, r\rangle$. Moreover, for all i , r_i does not occur free within P_i or T' . It follows that \mathcal{P} is an $R^!$ -confluent system.

To obtain an (M^-, R^+) -ready partition of \mathcal{P} we let $\{\mathcal{P}^{\tilde{r}} \mid \tilde{r} \text{ a finite subset of } R\}$ be the partition of \mathcal{P} defined by setting $Q \in \mathcal{P}^{\tilde{r}}$ if \tilde{r} are the free names of Q of sort R . So given T as above, $\tilde{r} = r_1 \dots r_n$. We can check that this partition satisfies the tidiness requirements:

- If $Q \in \mathcal{P}^{\tilde{r}}$ and $Q \xrightarrow{\alpha} Q'$ where $\alpha \notin M^- \cup R^+$ then $\alpha = \tau, n_G\langle\beta\rangle, \text{in}\langle x\rangle, \overline{\text{out}}\langle x\rangle, (\nu\tilde{y})\bar{c}\langle\tilde{x}, r\rangle, \bar{r}\langle y\rangle$, where $c : M_n^m$ and $r : R_T^\mu, \mu \neq \kappa$. It is direct to see that this transition does not affect the R -names of Q and $Q' \in \mathcal{P}^{\tilde{r}}$.
- If $Q \in \mathcal{P}^{\tilde{r}}$ and $Q \xrightarrow{\alpha} Q'$ where $\alpha \in M^-$, and $\alpha = (\nu\tilde{y}r)\bar{a} * \bar{m}\langle\tilde{x}, r\rangle$, then r may easily be seen to occur free and unguarded within Q' only in positive subject position. Thus $Q' \in \mathcal{P}^{\tilde{r}, r}$ and Q' is of the form T above. Note that $Q \not\xrightarrow{r}$ where $\alpha = (\nu\tilde{y})\bar{a} * \bar{m}\langle\tilde{x}, r\rangle, r \in \text{fn}(Q)$ since as illustrated above, for all $r : R \in \text{fn}(Q)$, r is not controlled in Q .
- If $Q \in \mathcal{P}^{\tilde{r}}$ and $Q \xrightarrow{\alpha} Q'$ where $\alpha \in R^+$ then it must be that $\text{subj}(\alpha) = r$ for some $r \in \tilde{r}$ and since r does not occur in Q' , $Q' \in \mathcal{P}^{\tilde{r}-r}$.

So the partition is (M^-, R^+) -tidy. Also \mathcal{P} is (M^-, R^+) -ready since for every $Q \in \mathcal{R}^{\tilde{r}}$ it is clear that Q can perform every $r \in \tilde{r}$. \square

Lemma 6.6.11 \mathcal{I}^R is (M^+, R^-) -disciplined.

PROOF: Let $Q \in \mathcal{I}^R$. Then Q is a derivative of I^R and so by Corollary 6.6.5, Q is fully Π -confluent which implies that Q is $R^!$ -confluent as required.

To obtain an (M^+, R^-) -tidy partition of \mathcal{I}^R we take $\{\mathcal{I}^{\tilde{r}} \mid \tilde{r} \text{ a finite subset of } R\}$ where $Q \in \mathcal{I}^{\tilde{r}}$ if \tilde{r} are the names of sort R occurring free in Q . This partition is (M^+, R^-) -tidy as we can check that

- If $Q \in \mathcal{I}^{\tilde{r}}$ and $Q \xrightarrow{\alpha} Q'$ where $\alpha \notin M^+ \cup R^-$, then from the definitions of I and I' we see that α must be τ or $(\nu\gamma)n_G\langle\gamma\rangle$. It is straightforward to verify that the free names of sort R do not change as a result of such a move and so $Q' \in \mathcal{I}^{\tilde{r}}$.
- If $Q \in \mathcal{I}^{\tilde{r}}$ and $Q \xrightarrow{\alpha} Q'$ where $\alpha \in R^-$, then certainly r must occur free in Q and so $r \in \tilde{r}$. Moreover, by Lemma 6.4.9(2)(b), r does not occur free in Q' thus $Q' \in \mathcal{I}^{\tilde{r}-r}$.
- If $Q \in \mathcal{I}^{\tilde{r}}$ and $Q \xrightarrow{\alpha} Q'$ where $\alpha \in M^+$ and $\text{obj}_R(\alpha) \notin \text{fn}(Q)$, then it is easy to see that r occurs free in Q' and hence $Q' \in \mathcal{I}^{\tilde{r}, r}$ as required.

Moreover, \mathcal{I} is (M^+, R^-) -disciplined as if $Q \in \mathcal{I}'$ and $Q \downarrow$, then by Lemma 6.6.8, $Q \Rightarrow \xrightarrow{\bar{r}(v)}$ for some v . \square

It now remains to prove that $I^b \simeq_{\downarrow}^{R^-} I^b$. The proof consists a careful analysis of the behaviour of the systems which is facilitated by the fact that they are fully Π -confluent. We begin by considering the first transformation.

6.6.4 Transformation 1

Let Q be a derivative of agent I^b where all objects are in the quiescent state and suppose that Q receives an invocation for some method m in one of its G -objects, α . Various possibilities exist:

- α may return the result of the invocation to the environment, or
- it may invoke a method in another object β within the society, or
- it may delegate responsibility for returning the result to another society object γ .

In the first case, since the society no longer owes answers to the environment, it may accept new method invocations (recall that we are dealing with I^b , the fragment of the transition system of I where at most one answer may be outstanding at any point). If α is capable of further activity, this may lead to the existence of more than one threads of activity within the society. In the second and third cases, no new methods may be invoked within the society. In the second case the activity of α is suspended until a result is returned while control is passed to object β which may subsequently behave in a similar manner. Unlike the other two cases α retains the responsibility for returning the result of method m . In the third case, α frees itself from this task and it may thus continue with further activity. Moreover, a second object is activated, namely γ which also assumes the responsibility of answering method m .

In this way Q may evolve to a system containing a number of active objects each of which associated with a tail of objects suspended and waiting for a result to a method invocation as shown in Figure 6.2. We will call an active object and its associated tail a *chain* and we may observe that at most one of these chains may be able to return a result to a method originating from outside the society, at most one since the system is a derivative of I^b . The exact format of derivatives of Q will be given formally later.

We begin with some process-calculus definitions.

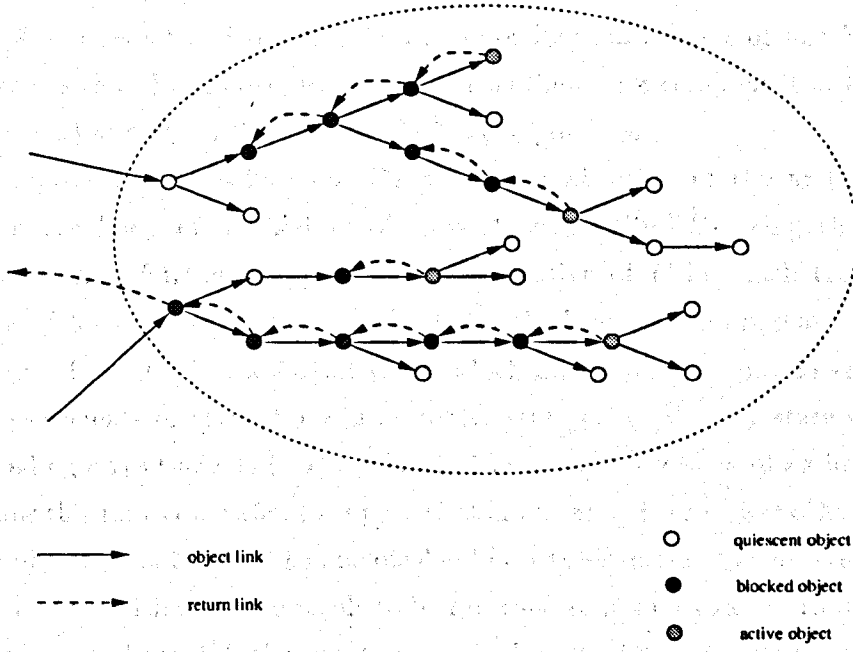


Figure 6.2: A society

Definition 6.6.12 A *chain* is an agent C of the form:

$$C \equiv (\nu \tilde{p})(P_1\langle r_1 \rangle \mid P_2\langle r_1, r_2 \rangle \mid \dots \mid P_k\langle r_{k-1} \rangle)$$

where

1. C is R-closed and $n(C) \upharpoonright R = r_1 \dots r_{k-1}$,
2. r_i is uniquely borne by P_i and uniquely handled by P_{i+1} for all i , and
3. r_i occurs unguarded in P_i and if $P_i \xrightarrow{\alpha}$ then $\text{subj}(\alpha) = r_i$.

Similarly, an *r-chain* is an agent of the form:

$$C\langle r_0 \rangle \equiv (\nu \tilde{p})(P_1\langle r_0, r_1 \rangle \mid P_2\langle r_1, r_2 \rangle \mid \dots \mid P_k\langle r_{k-1} \rangle)$$

1. $n(C) \upharpoonright R = r_0 \dots r_{k-1}$, $\text{fn}(C) \upharpoonright R = \{r_0\}$,
2. r_i is uniquely borne by P_i and uniquely handled by P_{i+1} for all i , and
3. r_i occurs unguarded in P_i and if $P_i \xrightarrow{\alpha}$ then $\text{subj}(\alpha) = r_i$.

Thus the distinction between a chain and an *r-chain* is that while a chain possesses no free R-names (representing a thread of activity initiated within the society), an

r -chain possesses a single free R -name (representing a thread of activity originating from the environment and responsible for responding via a name of sort R). For a chain (resp. r -chain) as above, we refer to P_k as the *active* component of the chain (resp. r -chain) and $P_1 \dots P_{k-1}$ as the *blocked* components.

Suppose $Pdec'$ results from $Pdec$ by an application of the first transformation in the body of method m of class A , $\text{return } E!m'(\tilde{E})$ being changed to $\text{commit } E!m'(\tilde{E})$. Further, suppose P is a derivative of I^b in which the name r is to be used for the return of the result of a method invocation originating outside the society. Then I'^b has a derivative P' which differs from P only in that in P' some object-agents of class A are in an active state or a quiescent state while the corresponding object-agents in P are blocked awaiting the return of an invocation. To see how this may come about suppose that in reaching P an object α has invoked m in an object β and that β has invoked m' (via the command in question) in an object γ and is waiting for the result to be returned to it so it can return it to α . In the computation from I'^b , the agent corresponding to β would have committed to γ the responsibility for returning a result to α , thus freeing itself to continue with other activity. Figure 6.3 gives an example of a chain in P and below it the state of the corresponding objects in P' .

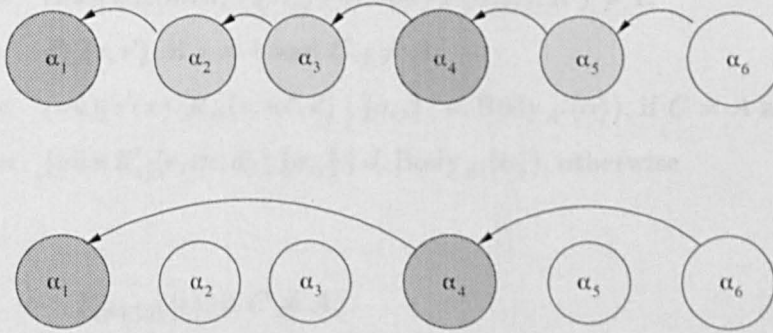


Figure 6.3: A chain and a t-chain

This motivates the following definition.

Definition 6.6.13 Let A be a class and A' the class obtained after a single application of transformation 1 in the body S of method m of class A . An *object chain* is a chain C of the form

$$\begin{aligned}
 C \equiv & (\nu \tilde{p})(P_{11}\langle r_{11} \rangle \mid P_{12}\langle r_{11}, r_{12} \rangle \mid \dots \mid P_{1k_1}\langle r_{1(k_1-1)}, r_{1k_1} \rangle \\
 & \mid P_{21}\langle r_{1k_1}, r_{21} \rangle \mid P_{22}\langle r_{21}, r_{22} \rangle \mid \dots \mid P_{2k_2}\langle r_{2(k_2-1)}, r_{2k_2} \rangle \\
 & \mid \dots
 \end{aligned}$$

$$| P_{n1}\langle r_{(n-1)k_{n-1}}, r_{n1} \rangle | P_{n2}\langle r_{n1}, r_{n2} \rangle | \dots | P_{nk_n}\langle r_{n(k_n-1)}, r_{nk_n} \rangle \\ | P_{(n+1)1}\langle r_{nk_n} \rangle$$

where

$$\begin{aligned} P_{ij}\langle r, r' \rangle &= (\nu \tilde{u})(r'(x). \bar{r}\langle x \rangle. Q_{ij}\langle m, d \rangle | [\sigma_{ij}] | d. \text{Body}_A\langle \alpha_{ij} \rangle), \text{ if } j \neq 1 \\ &= (\nu \tilde{u})(r'(x). R_{ij}\langle r, m', d \rangle | [\sigma_{ij}] | d. \text{Body}_{C_{ij}}\langle \alpha_{ij} \rangle), \text{ otherwise} \\ P_{(n+1)1}\langle r \rangle &= (\nu \tilde{u})(R_{(n+1)1}\langle r, m', d \rangle | [\sigma_{(n+1)1}] | d. \text{Body}_C\langle \alpha_{(n+1)1} \rangle) \end{aligned}$$

and either $C_{ij} \neq A$, or $m \neq m'$ or $R_{ij} \not\vdash \bar{r}\langle x \rangle. R'_{ij}$ for any R'_{ij} .

The $t_A^{A'}$ -chain C^T corresponding to an object chain C is given as follows:

$$\begin{aligned} C^T &= (\nu \tilde{r})(P'_{11}\langle r_1 \rangle | P'_{12} | \dots | P'_{1k_1}) \\ &\quad | P'_{21}\langle r_1, r_2 \rangle | P'_{22} | \dots | P'_{2k_2} \\ &\quad | \dots \\ &\quad | P'_{n1}\langle r_{n-1}, r_n \rangle | P'_{n2} \dots | P'_{nk_n} \\ &\quad | P'_{(n+1)1}\langle r_n \rangle \end{aligned}$$

where for all i, j ,

$$\begin{aligned} P'_{ij}\langle r, r' \rangle &= (\nu \tilde{u})(Q_{ij}\langle m, d \rangle | [\sigma_{ij}] | d. \text{Body}_{A'}\langle \alpha_{ij} \rangle), \text{ if } j \neq 1, \\ &= P_{ij}\langle r, r' \rangle, \text{ if } j = 1 \text{ and } C_{ij} \neq A, \\ &= (\nu \tilde{u})(r'(x). R_{ij}\langle r, m', d \rangle | [\sigma_{ij}] | d. \text{Body}_{A'}\langle \alpha \rangle), \text{ if } C = A \text{ and } m' \neq m \\ &= (\nu \tilde{u})(R'_{ij}\langle r, m, d \rangle | [\sigma_{ij}] | d. \text{Body}_{A'}\langle \alpha \rangle), \text{ otherwise} \end{aligned}$$

and

$$\begin{aligned} P'_{(n+1)1}\langle r \rangle &= P_{(n+1)1}\langle r \rangle, \text{ if } C \neq A, \\ &= (\nu \tilde{u})(R_{(n+1)1}\langle r, m', d \rangle | [\sigma] | d. \text{Body}_{A'}\langle \alpha \rangle), \text{ if } C = A \text{ and } m' \neq m \\ &= (\nu \tilde{u})(R'_{(n+1)1}\langle r, m, d \rangle | [\sigma] | d. \text{Body}_{A'}\langle \alpha \rangle), \text{ otherwise} \end{aligned}$$

where

$$\begin{aligned} R'_{ij}\langle r, m, d \rangle &= C[[\text{commit } E!m'(\tilde{E})]], \text{ if } R_{ij}\langle r, m, d \rangle = C[[\text{return } E!m'(\tilde{E})]] \\ &= (\nu \tilde{v}, \tilde{d}, d')(D\langle \tilde{v}, \tilde{d} \rangle | v_i(x_i). \dots \bar{d}_n. v_n(x_n). \bar{x}_0\langle \tilde{x}, r \rangle. \bar{d}' | d'. R\langle d \rangle), \\ &\quad \text{if } R_{ij}\langle r, d \rangle = (\nu \tilde{v}, \tilde{d}, d')(D\langle \tilde{v}, \tilde{d} \rangle \\ &\quad \quad | v_i(x_i). \dots \bar{d}_n. v_n(x_n). (\nu r')\bar{x}_0\langle \tilde{x}, r' \rangle. r'(x). \bar{r}\langle x \rangle. \bar{d}') \\ &\quad \quad | d'. R\langle d \rangle) \\ &= R_{ij}\langle r, m, d \rangle, \text{ otherwise} \end{aligned}$$

An object r -chain and its corresponding $t_A^{A'}$ -chain is defined in a similar way. \square

So given a chain C the corresponding $t_A^{A'}$ -chain may be obtained by updating the connection of the object-components to reflect the effect of the transformation and replacing all agents representing A objects by appropriate agents representing A' objects. Note that in the definition above we have used the observation that, given an object of class A executing method m ,

$$P\langle r, r' \rangle = (\nu \tilde{u})(r'(x). R\langle r, m, d \rangle \mid [\sigma] \mid d. \text{Body}_{C_A}(\alpha)),$$

the following possibilities exist for component $R\langle r, m, d \rangle$:

- $R\langle r, m, d \rangle = C[[\text{return } E!m'(\tilde{E})]]$ for $C[\cdot]$ a command context, corresponding to a state where execution of $\text{return } E!m'(\tilde{E})$ has not yet been initiated, or
- $R\langle r, m, d \rangle = (\nu \tilde{v}, \tilde{d}, d')(D\langle \tilde{v}, \tilde{d} \rangle \mid v_i(x_i). \dots \overline{d_n}. v_n(x_n). (\nu r') \overline{x_0}(\tilde{x}, r'). r'(x). \bar{r}(x). \overline{d'})$ corresponding to a state where the command is in the process of being executed, and
- $R\langle r, m, d \rangle = P$ where $r \notin \text{fn}(P)$, corresponding to a state where execution of $\text{return } E!m'(\tilde{E})$ has been completed.

It is easy to see that a t -chain may mimic the behaviour of its respective chain very closely. This is captured by the following lemma:

Lemma 6.6.14 Suppose C is an object chain with $t_A^{A'}$ -chain C^T . If $C \xrightarrow{\alpha} S$ then $C^T \xrightarrow{\beta} R$ where

- $\alpha = \beta$ or $\alpha = (\nu r) \bar{a}(\tilde{x}, r)$ and $\beta = \bar{a}(\tilde{x}, r')$, and
- if $\alpha = \tau$ then either $S^T = R$ or $S \implies S' = (\Pi C_i \mid C')$ and $R = (\Pi C_i^T \mid C'^T)$, for some chains C', C_i .

PROOF: The proof relies on the definition of a $t_A^{A'}$ -chain. For suppose $C \xrightarrow{\alpha} S$ where

$$P\langle r' \rangle = (\nu \tilde{p})(R\langle r', m', d \rangle \mid [\sigma] \mid d. \text{Body}_C(\alpha))$$

is the active component of C . Consider $C^T\langle r \rangle$ and let $P'\langle r' \rangle$ be the active object of C^T . The following possibilities exist:

- $C \neq A$ and $P'\langle r' \rangle = P\langle r' \rangle$.
- $C = A$ and $m \neq m'$. Then $P'\langle r' \rangle = (\nu \tilde{p})(R\langle r', m', d \rangle \mid [\sigma] \mid d. \text{Body}_{A'}(\alpha))$.

- $C = A$, $m' = m$ and $R\langle r', m, d \rangle = C[[\text{return } E!m'(\tilde{E})]]$ for some command context C . Then $P'\langle r' \rangle = (\nu \tilde{p})(C[[\text{commit } E!m'(\tilde{E})]]\langle r', d \rangle \mid [\sigma] \mid d. \text{Body}_{A'}\langle \alpha \rangle)$.
- Suppose $C = A$, $m' = m$ and

$$R\langle r', m, d \rangle = (\nu \tilde{v} \tilde{d})(D\langle \tilde{v}, \tilde{d} \rangle \mid v_i(x_i) \dots (\nu r'') \overline{x_0}(\tilde{x}, r''). r''(x). \overline{r'}(x). \overline{d'}) \mid d'. U\langle d \rangle)$$

Then

$$P'\langle r' \rangle = (\nu \tilde{p} \tilde{v} \tilde{d})(D\langle \tilde{v}, \tilde{d} \rangle \mid v_i(x_i) \dots \overline{x_0}(\tilde{x}, r'). \overline{d'}) \mid d'. U\langle d \rangle \mid [\sigma] \mid d. \text{Body}_{A'}\langle \alpha \rangle)$$

It is easy to see in each of these cases that actions of C can be matched by actions of C^T . In particular, the last case explains how an action $\alpha = (\nu r'') \overline{a}(\tilde{x}, r'')$ of C may be matched by an action $\beta = \overline{a}(\tilde{x}, r')$ by C^T . In addition, if $C\langle r \rangle \xrightarrow{\tau} S$ then $C^T\langle r \rangle \xrightarrow{\tau} S'$ and $S^T = S'$ with one exception: Suppose

$$C\langle r \rangle = (\nu \tilde{q})(E\langle r, r_0 \rangle \mid R\langle r_0, r_1 \rangle \mid P_1\langle r_1, r_2 \rangle \mid P_2\langle r_2, r_3 \rangle \mid \dots \mid P_n\langle r_n, r' \rangle \mid P\langle r' \rangle) \xrightarrow{\tau} C'\langle r \rangle = (\nu \tilde{q})(E\langle r, r_0 \rangle \mid R\langle r_0, r_1 \rangle \mid P_1\langle r_1, r_2 \rangle \mid P_2\langle r_2, r_3 \rangle \mid \dots \mid Q_n\langle r_n \rangle \mid Q)$$

where

$$P_i\langle r_i, r_{i+1} \rangle = (\nu \tilde{q}, d)(r_{i+1}(x), \overline{r_i}(x). Q_i\langle m, d \rangle \mid [\sigma_i] \mid d. \text{Body}_A\langle \alpha_i \rangle),$$

$$P\langle r' \rangle \xrightarrow{\overline{r'}(x)} Q, \text{ and}$$

$$P_n\langle r_n, r' \rangle \xrightarrow{r'(x)} Q_n\langle r_n \rangle.$$

Note that $C^T \not\Rightarrow C'^T$. Nonetheless, $C'\langle r \rangle$ may engage in $n + 1$ silent transitions to give

$$C'\langle r \rangle \Rightarrow S = (\nu \tilde{q})(E\langle r, r_0 \rangle \mid R'\langle r_0 \rangle \mid P'_1 \mid \dots \mid P'_n \mid Q)$$

where $P'_i = (\nu \tilde{q}, d)(Q_i\langle m, d \rangle \mid [\sigma_i] \mid d. \text{Body}_A\langle \alpha_i \rangle)$. Thus S is a parallel composition of $n + 2$ object chains, namely $(\nu \tilde{q})(E\langle r, r_0 \rangle \mid R\langle r_0 \rangle)$, and P'_i for all i and Q .

By the definition of a t-chain,

$$C^T\langle r \rangle = (\nu \tilde{q})(E'\langle r, r_0 \rangle \mid T\langle r_0, r' \rangle \mid P''_1 \mid P''_2 \mid \dots \mid P''_n \mid P'\langle r' \rangle) \xrightarrow{\tau} S'\langle r \rangle = (\nu \tilde{q})(E'\langle r, r_0 \rangle \mid T'\langle r_0 \rangle \mid P''_1 \mid P''_2 \mid \dots \mid P''_n \mid Q')$$

where

$$P''_i = (\nu \tilde{q}, d)(Q_i\langle m, d \rangle \mid [\sigma_i] \mid d. \text{Body}_{A'}\langle \alpha_i \rangle),$$

$$T\langle r_0, r' \rangle \xrightarrow{r(x)} T'\langle r_0 \rangle, \text{ and}$$

$$P'\langle r' \rangle \xrightarrow{\bar{r}'(x)} Q'.$$

By close observation of the agents we deduce that

$$(\nu \tilde{q})(E\langle r, r_0 \rangle \mid R\langle r_0 \rangle)^T = (\nu \tilde{q})(E'\langle r, r_0 \rangle \mid T'),$$

$$P_i^T = P_i'', \text{ and}$$

$$Q^T = Q' \text{ as required.} \quad \square$$

We may now state our main result.

Theorem 6.6.15 $I^b \simeq_1^{R^-} I^b.$

PROOF: The proof achieves a close correspondence between each derivative of I^b with a derivative of I^b . Of course, it is the case that I^b is capable of activity which can not be mirrored directly by I^b : for example an object executing the transformed method within I^b will delegate responsibility of returning to some other object and subsequently, possibly invoke another method or enter the quiescent state and have one of its methods invoked, unlike the respective object within I^b which will be blocked awaiting the return of the call. However, the fact of the system's confluence allows us to conclude the required results.

Suppose that the transformation has been applied on the body S of method m_1 in class A_1 . The proof relies on the observation that any computation of process I^b can be very closely mimicked by process I^b . This correspondence is captured by the following relation.

$$\begin{aligned} \mathcal{R} = \{ & (S_1, S_2) \mid S_1 = (\nu \tilde{p})(C\langle r \rangle \mid \prod_{i \in I} C_i \mid \prod_{j \in J} \text{Obj}_{A_1} \langle \alpha_j \rangle \mid \prod_{k \in K} \text{Obj}_c \langle \alpha_k \rangle \mid I), \\ & S_2 = (\nu \tilde{p})(C^T\langle r \rangle \mid \prod_{i \in I} C_i^T \mid \prod_{j \in J} \text{Obj}_{A_1'} \langle \alpha_j \rangle \mid \prod_{k \in K} \text{Obj}_c \langle \alpha_k \rangle \mid I'), \\ & \text{where } c \in \{G, A_2 \dots A_n\} \\ & C\langle r \rangle \text{ is an object } r\text{-chain and for all } i, C_i \text{ is an object chain,} \\ & C^T\langle r \rangle, C_i^T \text{ are the corresponding } t_{A_1'}^{A_1}\text{-chains,} \\ & I^b \xRightarrow{s} S_1 \text{ and } I^b \xRightarrow{s} S_2 \text{ for some } s \} \end{aligned}$$

Thus a derivative of I^b is a parallel composition containing an r -chain $C\langle r \rangle$ (which may be empty), a number of chains and a number of G and \tilde{A} object-agents in the quiescent state. Such a derivative is related by \mathcal{R} to a derivative of I^b composed of the corresponding $t_{A_1'}^{A_1}$ chains and G, A_1', A_2, \dots, A_n object-agents. Suppose $(S_1, S_2) \in \mathcal{R}$. We observe that the following holds:

If $S_1 = (\nu\tilde{p})(\text{Obj}_C(\beta) \mid \dots)$ then $S_2 = (\nu\tilde{p})(\text{Obj}_{C'}(\beta) \mid \dots)$, where $C' = A'_1$ if $C = A_1$ and $C' = C$, otherwise.

The proof of the result follows directly from the construction of \mathcal{R} . Note that by definition, an object chain contains no object components in the quiescent state.

We may prove the following properties.

Properties 6.6.16 Let $(S_1, S_2) \in \mathcal{R}$. The following hold:

1. if $S_1 \xrightarrow{\alpha} S'_1$ then $S_2 \xrightarrow{\alpha} S'_2$ and either $(S'_1, S'_2) \in \mathcal{R}$, or $\alpha = \tau$ and $S'_1 \Rightarrow S''_1$ with $(S''_1, S'_2) \in \mathcal{R}$;
2. $S_1 \downarrow R^-$ iff $S_2 \downarrow R^-$;

PROOF: The proof of the first Property is a case analysis on $S_1 \xrightarrow{\alpha} S'_1$. The following possibilities exist.

- $\alpha = a * m(\tilde{z}, r)$ and $S_1 = (\nu\tilde{p})(\text{Obj}_G(a) \mid \dots) \xrightarrow{\alpha} S'_1 = (\nu\tilde{p})(C(r) \mid \dots)$ where $\text{Obj}_G(a) \xrightarrow{\alpha} C(r) = (\nu\tilde{q}r)(Q(r, d) \mid [\sigma] \mid d.\text{Body}_G(a))$. By an earlier observation,

$$S_2 = (\nu\tilde{p})(\text{Obj}_G(a) \mid \dots) \xrightarrow{\alpha} S'_2 = (\nu\tilde{p})(C(r) \mid \dots).$$

By definition $C(r) = C^T(r)$ and hence $(S'_1, S'_2) \in \mathcal{R}$.

- $\alpha = (\nu\gamma)\bar{n}(\gamma)$ and $S_1 = (\nu\tilde{p})P \xrightarrow{\alpha} S'_1 = (\nu\tilde{p})(P \mid \text{Obj}_G(\gamma))$. Then $S_2 = (\nu\tilde{p})P' \xrightarrow{\alpha} S'_2 = (\nu\tilde{p})(P' \mid \text{Obj}_G(\gamma))$. Clearly, $(S'_1, S'_2) \in \mathcal{R}$.
- $\alpha = \bar{r}(v)$, where $S_1 = (\nu\tilde{p})(C(r) \mid D) \xrightarrow{\alpha} S'_1 = (\nu\tilde{p})(S \mid D)$ and $C(r) \xrightarrow{\alpha} S$. Note that by Lemma 6.4.8(3) $r \notin \text{fn}(S'_1)$. By the definition of an r-chain, $C(r)$ contains a single object component of the form $P(r) = (\nu\tilde{q})(Q(r, d) \mid [\sigma] \mid d.\text{Body}_C(a))$. By Lemma 6.6.14, $C^T(r) \xrightarrow{\alpha} S'$ where $S^T = S'$. Thus $S_2 \xrightarrow{\alpha} S'_2$ with $(S'_1, S'_2) \in \mathcal{R}$ as required.
- $\alpha = \tau$. Several possibilities exist depending on whether the transition is a transition internal to a chain, or an object creation or a method invocation initiated by an active object of a chain. In the former case the result is guaranteed by Lemma 6.6.14. Method invocations and object creations are similar to the first and second cases considered in this proof. Note that a method invocation by a chain results in increasing the length of the chain by one.

This completes the proof of the first property.

To prove the second property, suppose $S_1 \downarrow R^-$. Since S_1 is a derivative of I^b there are two cases to consider. First suppose that there is no $r : R$ occurring free in S_1 . Then by Lemma 6.6.8, $S_1 \Rightarrow S'_1$ where each object component of S'_1 is in the quiescent state. If $S_1 \not\vdash$ then by the same lemma each object component of S_1 is in the quiescent state and by the construction of \mathcal{R} , each component of S_2 is in the quiescent state. Thus $S_2 \downarrow$. So suppose $S_1 \xrightarrow{r} S_{11}$. By Property 6.6.16(1), $S_2 \xrightarrow{r} S_{21}$, where $S_{11} \Rightarrow S'_{11}$ and $(S'_{11}, S_{21}) \in \mathcal{R}$. By Π -confluence and since $S_1 \downarrow$, $S_1 \simeq_1 S'_{11}$ so $S'_{11} \downarrow R^-$. Repeated application of this argument gives that there exist S_{1i}, S'_{1i}, S_{2i} such that

$$\begin{array}{ccccccc} S_1 & \xrightarrow{r} & S_{11} & \Rightarrow & S'_{11} & \xrightarrow{r} & \dots & S'_{1(k-1)} & \xrightarrow{r} & S_{1k} & \Rightarrow & S'_{1k} \\ S_2 & \xrightarrow{r} & S_{21} & \xrightarrow{r} & \dots & S_{1(k-1)} & \xrightarrow{r} & S_{2k} \end{array}$$

where $(S'_{1i}, S_{2i}) \in \mathcal{R}$ and since $S_1 \downarrow$, $S'_{1k} \not\vdash$. By Lemma 6.6.8, each object component of S'_{1k} is in the quiescent state. Hence, by the construction of \mathcal{R} , each component of S_{2k} is in the quiescent state. Therefore $S_2 \downarrow$. Clearly, $\text{fn}(S_1) = \text{fn}(S_2)$. Thus there is no $r : R$ such that $r \in \text{fn}(S_2)$ and so $S_2 \downarrow R^-$.

Next we consider the case where there exists $r : R$ occurring free in S_1 . By Lemma 6.6.8, $S_1 \xrightarrow{\bar{r}(v)} S'_1$ where each component of S'_1 is in the quiescent state. As before, there exist S_{1i}, S'_{1i}, S_{2i} such that

$$\begin{array}{ccccccc} S_1 & \xrightarrow{r} & S_{11} & \Rightarrow & S'_{11} & \xrightarrow{r} & \dots & S'_{1(m-1)} & \xrightarrow{r} & S_{1m} & \Rightarrow & S'_{1m} \\ S_2 & \xrightarrow{r} & S_{21} & \xrightarrow{r} & \dots & S_{1(m-1)} & \xrightarrow{r} & S_{2m} \end{array}$$

where $(S'_{1i}, S_{2i}) \in \mathcal{R}$ and $S'_{1m} \not\vdash$. By Π -confluence and since $S_1 \downarrow$, $S'_{1m} \simeq_1 S_1$. Hence $S'_{1m} \xrightarrow{\bar{r}(v)} Q_1$ where $Q_1 \simeq_1 S'_1$. In addition, by Property 6.6.16(1), $S_{2m} \xrightarrow{\bar{r}(v)} Q_2$ where $(Q_1, Q_2) \in \mathcal{R}$. Using the same argument as above we can find Q_{1i}, Q'_{1i}, Q_{2i} such that

$$\begin{array}{ccccccc} Q_1 & \xrightarrow{r} & Q_{11} & \Rightarrow & Q'_{11} & \xrightarrow{r} & \dots & Q'_{1(l-1)} & \xrightarrow{r} & Q_{1l} & \Rightarrow & Q'_{1l} \\ Q_2 & \xrightarrow{r} & Q_{21} & \xrightarrow{r} & \dots & Q_{1(l-1)} & \xrightarrow{r} & Q_{2l} \end{array}$$

where $(Q'_{1i}, Q_{2i}) \in \mathcal{R}$ and $Q'_{1l} \not\vdash$. By Lemma 6.6.8, each object component of Q'_{1l} is in the quiescent state. Hence by the construction of \mathcal{R} , each component of Q_{2l} is in the quiescent state. Therefore $Q_2 \downarrow$. Moreover, since S_2 is fully Π -confluent, $S_2 \downarrow \bar{r}(v)$. Recall that S_2 being a derivative of I^b , there is at most one name of sort R occurring free in S_2 , namely r . Suppose there exists u such that $S_2 \uparrow \bar{r}(u)$. Then by Π -confluence it must be that $S_2 \uparrow \bar{r}(v)\bar{r}(u)$, that is $S_2 \xrightarrow{\bar{r}(v)} S'_2 \xrightarrow{\bar{r}(u)} \uparrow$. However, by Π -confluence $S'_2 \simeq_1 Q_{2l}$ and since $r \notin \text{fn}(Q_{2l})$ this gives a contradiction. Thus $S_2 \downarrow \rho$

for all $\rho \in R^-$, $\text{subj}(\rho) = r$, and since r is the only name of sort R occurring free in S_2 , $S_2 \downarrow R^-$.

To prove the converse, suppose $S_1 \uparrow R^-$. In addition, suppose $S_1 \xrightarrow{\tau} S_{11}$. Then by Property 6.6.16(1), $S_2 \xrightarrow{\tau} S_{21}$ and $S_{11} \Rightarrow S'_{11}$ where $(S'_{11}, S_{21}) \in \mathcal{R}$. By full Π -confluence, $S'_{11} \uparrow R^-$. Similarly, if $S_1 \xrightarrow{\bar{r}(v)} S'_1$ for some $r : R$ then $S_2 \xrightarrow{\bar{r}(v)} S'_2$ where $(S'_1, S'_2) \in \mathcal{R}$ and by full Π -confluence, $S'_1 \uparrow$. Repeated application of this argument shows that S_2 can closely mimic any finite or infinite computation of S_1 giving that $S_2 \uparrow R^-$ as required. \square

We may now proceed with the proof of the main theorem. Let \mathcal{B} be the relation $\simeq_1 \mathcal{R} \simeq_1$. We claim that $\mathcal{B} \subseteq \simeq_1^{R^-}$. Suppose $(S_1, S_2) \in \mathcal{B}$ and let S, S' be such that $(S, S') \in \mathcal{R}$, $S_1 \simeq_1 S$ and $S' \simeq_1 S_2$. Suppose $S_1 \downarrow R^-$. Then $S \downarrow R^-$ and by Property 6.6.16(2), $S' \downarrow R^-$. So $S_2 \downarrow R^-$. Similarly, if $S_2 \downarrow R^-$ then $S' \downarrow R^-$, $S \downarrow R^-$ and $S_1 \downarrow R^-$.

Suppose that $S_1 \downarrow R^-$ and $S_1 \xrightarrow{\alpha} S'_1$. Since $S \downarrow$, by Lemma 6.6.8 there exist Q, Q' such that $S \Rightarrow Q$, $S' \Rightarrow Q'$, $Q, Q' \not\rightarrow$ and $(Q, Q') \in \mathcal{R}$. Moreover by Π -confluence $S_1 \simeq_1 Q$ and $Q' \simeq_1 S_2$.

- Suppose $\alpha = \tau$. Then by Π -confluence $S_1 \simeq_1 S'_1 \simeq_1 Q$ and $(S'_1, S_2) \in \mathcal{R}$.
- Suppose $\alpha \neq \tau$. If $S_1 \downarrow \alpha$ then since $S_1 \simeq_1 Q$, $Q \xrightarrow{\alpha} R \simeq_1 S'_1$. By Property 6.6.16(1), $Q' \xrightarrow{\alpha} R'$ and $(R, R') \in \mathcal{R}$. There are two cases: if $Q' \downarrow \alpha$ then since $S_2 \simeq_1 Q'$, $S_2 \Rightarrow \xrightarrow{\alpha} S'_2 \simeq_1 R'$ with $(S'_1, S'_2) \in \mathcal{B}$ as required. Otherwise, if $Q' \uparrow \alpha$ then $S_2 \uparrow \alpha$ and by full Π -confluence $R' \uparrow$. Since additionally $S_2 \downarrow$, $S_2 \Rightarrow S'_2 \xrightarrow{\alpha} S'_2 \uparrow$ and $S'_2 \simeq_1 R'$. Hence $(S'_1, S'_2) \in \mathcal{B}$. The case $S_1 \uparrow \alpha$ is similar.

So suppose $S_2 \downarrow R$, $S_2 \xrightarrow{\alpha} S'_2$ and pick Q, Q' where $S_1 \simeq_1 Q$, $Q \mathcal{R} Q'$, $Q' \simeq_1 S_2$ such that $Q \not\rightarrow$. Several possibilities exist. We consider the case $S_1 \downarrow \alpha$, $S_2 \downarrow \alpha$. The other cases use similar ideas.

- Suppose $\alpha = \tau$. By Π -confluence and since $S_2 \downarrow$, $S_2 \simeq_1 S'_2$ and hence $(S_1, S'_2) \in \mathcal{R}$;
- Suppose $\alpha = (\nu\gamma)\overline{nG}\langle\gamma\rangle$. Since $S_2 \simeq_1 Q'$, $Q' \Rightarrow \xrightarrow{\alpha} T' \simeq_1 S'_2$. Moreover, by the construction of \mathcal{R} , $Q \xrightarrow{\alpha} R$ so by Property 6.6.16(1), $Q' \xrightarrow{\alpha} R'$ where $(R, R') \in \mathcal{R}$. By full Π -confluence $R' \simeq_1 T'$. Furthermore, $S_1 \simeq_1 Q$, so $S_1 \Rightarrow \xrightarrow{\alpha} S'_1 \simeq_1 R$ and clearly, $(S'_1, S'_2) \in \mathcal{R}$.
- Suppose $\alpha = a * m(\tilde{x}, r)$. Since $S_2 \simeq_1 Q'$, $Q' \Rightarrow \xrightarrow{\alpha} T'$. Moreover, since Q' is a derivative of I^b , there exist no names of sort R occurring free in Q' . By

- construction, $\text{fn}(Q) = \text{fn}(Q')$ so no names of sort R occur free in Q . Since Q is such that $Q \not\rightarrow$, by Lemma 6.6.8, every object-component of Q , and hence of Q' , is in the quiescent state. So $Q' \xrightarrow{\alpha} T''$, where $T'' \simeq_1 T'$ and $Q' = (\nu \tilde{p})(\text{Obj}_G\langle a \rangle \mid C)$ for some C . By construction of \mathcal{R} , $Q = (\nu \tilde{p})(\text{Obj}_G\langle a \rangle \mid C') \xrightarrow{\alpha} T$ where $(T, T'') \in \mathcal{R}$. Furthermore, by definition of \simeq_1 , $S_1 \xRightarrow{\alpha} S'_1 \simeq_1 T$ and hence $(S'_1, S'_2) \in \mathcal{B}$.
- Suppose $\alpha = \bar{r}\langle v \rangle$. Since $S_2 \simeq_1 Q'$, $Q' \xRightarrow{\alpha} T'$. Moreover, since $\text{fn}(Q) = \text{fn}(Q')$, r occurs free in Q and as $Q \downarrow R^-$, by Lemma 6.6.8, $Q \xrightarrow{\bar{r}\langle u \rangle} T$. By Property 6.6.16(1), $Q' \xrightarrow{\bar{r}\langle u \rangle} T''$ where $(T, T'') \in \mathcal{R}$ and by Π -confluence, and since $\text{fn}(T') \upharpoonright R = \emptyset$, $u = v$ and $T' \simeq_1 T''$. In addition, $S_1 \xRightarrow{\alpha} S'_1 \simeq_1 R$. Therefore, $(S'_1, S'_2) \in \mathcal{B}$ as required.

Hence $\mathcal{B} \subseteq \simeq_1^{R^-}$. Since $(I^b, I^b) \in \mathcal{B}$, this completes the proof of the theorem. \square

6.6.5 Transformation 2

In this section we consider the second transformation. The intuition behind the soundness of this transformation is based on the idea that since the commands S and return E do not share any variables then the order in which they take place within a method body of a society is immaterial. This intuition is captured formally with the aid of confluence.

We begin with an observation concerning the encoding of a subcommand of a command.

Observation 6.6.17 Suppose S' is a subcommand of S . Then $\llbracket S \rrbracket = \mathcal{C}[\llbracket S' \rrbracket]$, where if $\llbracket S \rrbracket = \llbracket S \rrbracket \langle d \rangle$ then

1. $\mathcal{C}[Q] = (\nu \tilde{p}, d_1, d_2)(P\langle d_1 \rangle \mid d_1.Q\langle d_2 \rangle \mid d_2.R\langle d \rangle)$ where $d_2 \notin \text{fn}(P)$, $d_1 \notin \text{fn}(R)$, or
2. $\mathcal{C}[Q] = (\nu \tilde{p}, d')(Q\langle d' \rangle \mid d'.R\langle d \rangle)$, or
3. $\mathcal{C}[Q] = (\nu \tilde{p}, d')(P\langle d' \rangle \mid d'.Q\langle d \rangle)$, or
4. $\mathcal{C}[Q] = (\nu \tilde{p})Q\langle d \rangle$.

for P and R such that if $P\langle d \rangle \xrightarrow{\alpha} \xrightarrow{\bar{d}} P'$ then $P' \simeq_1 0$.

PROOF: The proof of this observation is by induction on the depth, d , of the occurrence of S' within S . For the base case, $d = 0$ we have that $S' \equiv S$. Then clearly, $\llbracket S \rrbracket = \llbracket S' \rrbracket$ and Clause 4 is satisfied.

For the induction step we have four cases:

- $S \equiv S_1; S_2$ where S' is a subcommand of S_1 . By definition

$$[S] = (\nu d)([S_1]\langle d \rangle \mid d. [S_2]).$$

Moreover, by the induction hypothesis $[S_1] \equiv C[[S']]$. So

$$[S] = (\nu d)(C[[S']]\langle d \rangle \mid d. [S_2])$$

and if $C[[S']]$ is of form 1 or 3, then $[S]$ is of form 1, otherwise it is of form 2.

- $S \equiv S_1; S_2$ where S' is a subcommand of S_2 . This is similar to the previous case.
- $S \equiv \text{if } E \text{ then } S_1 \text{ else } S_2$ where S' is a subcommand of S_1 . By definition,

$$[S] = (\nu d_1, d_2)([E]\langle \text{val} \rangle \mid \text{val}(b). \text{cond}(b \triangleright \overline{d_1}. 0, \text{true} \triangleright \overline{d_2}. 0) \mid d_1. [S_1]\langle \widetilde{p_1}, d \rangle \mid d_2. [S_2]\langle \widetilde{p_2}, d \rangle).$$

By the induction hypothesis, $[S_1] \equiv C[[S']]$ so $[S] = (\nu d_1, d_2)(([E]\langle \text{val} \rangle \mid \text{val}(b). \text{cond}(b \triangleright \overline{d_1}. 0, \text{true} \triangleright \overline{d_2}. 0) \mid d_1. C[[S']]) \mid d_2. [S_2])$. Hence $[S]$ is a context of form 1, if $C[\cdot]$ is of form 1 or 2, and a context of form 3, otherwise.

- $S \equiv \text{if } E \text{ then } S_1 \text{ else } S_2$ where S' is a subcommand of S_2 . This is similar to the previous case. \square

So assume that the transformation has been applied on the body S of method m_1 in class A_1 . Suppose the encoding of class A_1 is given by:

$$[\text{class } A_1] = !(\nu \alpha) \overline{n_{A_1}} \langle \alpha \rangle. \text{Obj}_{A_1} \langle \alpha \rangle$$

where

$$\text{Obj}_{A_1} = (\alpha \tilde{n})(\nu \tilde{p})([\text{var } \tilde{X} : \tilde{T}]\langle \tilde{p} \rangle \mid \text{Body} \langle \alpha, \tilde{n}, \tilde{p} \rangle)$$

$$\begin{aligned} \text{Body} &= (\alpha \tilde{n} \tilde{p})(\Sigma_{i=1}^q \alpha * m_i(\tilde{v}, r). (\nu d \tilde{u}) \\ &\quad ((\Pi \text{REG} \langle r_i, d_i, w_i, v_i \rangle \mid [\text{var } \tilde{Z} : \tilde{U}]) \langle \tilde{u} \rangle \mid [S_i] \langle \tilde{n}, \tilde{p}, \tilde{u}, m_i, r, d \rangle) \\ &\quad \mid d. \text{Body} \langle \alpha, \tilde{p} \rangle) \end{aligned}$$

where by the observation above $[S_1] = C[[S'; \text{return } E]]$ for some context $C[\cdot]$. Let B be an agent differing from $[\text{class } A_1]$ in that the encodings of the commands S' and $\text{return } E$ within method m_1 are concurrently composed with one another:

$$B = !(\nu \alpha) \overline{n_{A_1}} \langle \alpha \rangle. \text{Obj}_B \langle \alpha \rangle$$

where

$$\begin{aligned} \text{Obj}_B &= (\alpha \tilde{n})(\nu \tilde{p})([\text{var } \tilde{X} : \tilde{T}](\tilde{p}) \mid \text{Body}'(\alpha, \tilde{n}, \tilde{p})) \\ \text{Body}' &= (\alpha \tilde{n} \tilde{p})(\Sigma_{i=1}^q \alpha * m_i(\tilde{v}, r).(\nu d \tilde{u}) \\ &\quad ((\Pi \text{REG}(r_i, t_i, w_i, v_i) \mid [\text{var } \tilde{Z} : \tilde{U}](\tilde{u}) \mid P_i(\tilde{n}, \tilde{p}, \tilde{u}, m_i, r, d)) \\ &\quad \mid d.\text{Body}(\alpha, \tilde{n}, \tilde{p})) \end{aligned}$$

where $P_i = [S_i]$, for $i \neq 1$, $P_1 = C[D](d)$ and

$$D(d) = (\nu d_1 d_2)([S'](d_1) \mid [\text{return } E](d_2) \mid d_1.d_2.\bar{d}).$$

The purpose of the agent $d_1.d_2.\bar{d}$ is to ensure that both S' and $\text{return } E$ terminate execution and indicate so by signalling on d_1, d_2 respectively, before any further computation is initiated. Let $H = (\nu n_{\tilde{A}})([G, A_2, \dots, A_n] \mid B)$. We claim the following:

Lemma 6.6.18 H^b is fully Π -confluent.

PROOF: In order to prove this we need to establish that agent B does not violate the unique bearing and unique controlling properties and does not give rise to non-confluent activity. The necessary analysis is very similar to those of Sections 6.4 and 6.6.1. The point of interest here is to prove the analogue of Lemma 6.4.7 for agent

$$Q \stackrel{\text{def}}{=} (\nu \tilde{z})([S_1] \mid [S_2] \mid [\sigma])$$

where S_1, S_2 occurs in the method body of a class of a community and S_1, S_2 share no common variables. We may see that there exist σ_1, σ_2 such that $\sigma = \sigma_1 \cup \sigma_2$ and

$$Q \equiv (\nu \tilde{z})([S_1] \mid [\sigma_1] \mid [S_2] \mid [\sigma_2])$$

where $\text{fn}([S_1] \mid [\sigma_1]) \cap \text{fn}([S_2] \mid [\sigma_2]) = \emptyset$. With this observation and since each of $(\nu \tilde{z})([S_1] \mid [\sigma_1])$ and $(\nu \tilde{z})([S_2] \mid [\sigma_2])$, satisfy the assumptions of Lemma 6.4.7, it is not difficult to prove that an execution of Q also satisfies the properties captured in that lemma. Thus, assuming Q is placed in an appropriate context, it will not be the first to violate conditions that guarantee absence of sharing.

The remaining arguments are similar to those of the proof of Corollary 6.6.5 and are omitted. \square

By the construction of H , it is now not difficult to see that for any computation of I (resp. I') there is an equivalent computation of H where execution of $[S']$ takes place before that of $[\text{return } E]$ (resp. vice versa). This observation

(which we prove formally below), along with the property of confluence which is satisfied by the agents involved, allows us to show that $I^b \simeq_1^{R^-} H^b$ and $I^b \simeq_1^{R^-} H^b$. This is the strategy adopted to prove the main theorem given below. Note that I' now denotes the encoding of the society described by I after a single application of transformation 2.

Theorem 6.6.19 $I^b \simeq_1^{R^-} I^b$.

PROOF: Recall that as defined above, D is the agent abstraction

$$D\langle d \rangle = (\nu d_1 d_2)([S']\langle d_1 \rangle \mid [\text{return } E]\langle d_2 \rangle \mid d_1. d_2. \bar{d}).$$

We define \mathcal{R} as follows:

$$\begin{aligned} \mathcal{R} = \{ (S_1^b, S_2^b) \mid & S_1 = (\nu \tilde{u})(\Pi_{i \in I} P_i \mid \Pi_{j \in J} Q_j \mid \Pi_{k \in K} W_k \mid \Pi_{l \in L} R_l \mid \Pi_{m \in M} T_m \mid I), \\ & S_2 = (\nu \tilde{u})(\Pi_{i \in I} P'_i \mid \Pi_{j \in J} Q'_j \mid \Pi_{k \in K} W'_k \mid \Pi_{l \in L} R'_l \mid \Pi_{m \in M} T_m \mid H), \\ & \text{where} \\ & P_i = (\nu \tilde{p})(C_i[[S'; \text{return } E]]\langle d \rangle \mid [\sigma_i] \mid d. \text{Body}\langle \alpha_i \rangle), \\ & P'_i = (\nu \tilde{p})(C_i[D]\langle d \rangle \mid [\sigma_i] \mid d. \text{Body}'\langle \alpha_i \rangle), \\ & \text{and for } F \text{ a derivative of } [S'], \\ & Q_j = (\nu \tilde{q} d_1 d_2)(F\langle d_1 \rangle \mid d_1. [\text{return } E]\langle d_2 \rangle \mid d_2. P\langle d \rangle \\ & \quad \mid [\sigma_j] \mid d. \text{Body}\langle \alpha_j \rangle), \\ & Q'_j = (\nu \tilde{q} d_1 d_2 d')(F\langle d_1 \rangle \mid [\text{return } E]\langle d_2 \rangle \mid d_1. d_2. \bar{d}' \mid d'. P\langle d \rangle \\ & \quad \mid [\sigma_j] \mid d. \text{Body}'\langle \alpha_j \rangle), \\ & \text{and for } V \text{ a derivative of } [\text{return } E], \\ & W_k = (\nu \tilde{v} d_2)(V\langle d_2 \rangle \mid d_2. P\langle d \rangle \mid [\sigma_k] \mid d. \text{Body}\langle \alpha_k \rangle), \\ & W'_k = (\nu \tilde{v} d_2 d')(V\langle d_2 \rangle \mid d_2. \bar{d}' \mid d'. P\langle d \rangle \mid [\sigma_k] \mid d. \text{Body}'\langle \alpha_k \rangle), \\ & R_l = (\nu \tilde{r})(E\langle d \rangle \mid [\sigma_l] \mid d. \text{Body}\langle \alpha_l \rangle), \\ & R'_l = (\nu \tilde{r})(E\langle d \rangle \mid [\sigma_l] \mid d. \text{Body}'\langle \alpha_l \rangle), \text{ for some } E \text{ and} \\ & I^b \xRightarrow{s} S_1^b, H^b \xRightarrow{s} S_2^b \text{ for some } s \} \end{aligned}$$

Note that an agent P_i represents an object executing method m_1 before it has reached the point where the transformation has been applied. Similarly, Q_j , W_k and R_l represent objects executing method m_1 but where the ‘interesting’ commands S' and $\text{return } E$ respectively, are in the process of being executed. Additionally, agents R_l represent objects of class A which are either executing method m_1 but have completed the commands that have undergone transformation, or executing a method other than m_1 , or are in the quiescent state. Finally, T_m represents objects of the remaining classes. The following properties hold:

Properties 6.6.20 Let $(S_1, S_2) \in \mathcal{R}$. Then

1. if $S_1 \xrightarrow{\alpha} S'_1$ then $S_2 \xrightarrow{\alpha} S'_2$ and $(S'_1, S'_2) \in \mathcal{R}$;
2. $S_1 \downarrow R^-$ iff $S_2 \downarrow R^-$.

PROOF: Let $(S_1, S_2) \in \mathcal{R}$. The first property is trivial to verify by the construction of \mathcal{R} . We consider a couple of cases:

- Suppose $S_1 = (\nu \tilde{u})(Q_j \mid C) \xrightarrow{\alpha} S'_1 = (\nu \tilde{u})(S'_j \mid C)$ where

$$\begin{aligned} Q_j &= (\nu \tilde{q}, d_1, d_2)(D\langle d_1 \rangle \mid d_1. [\text{return } E]\langle d_2 \rangle \mid d_2. P\langle d \rangle \mid [\sigma_j] \mid d. \text{Body}\langle \alpha_j \rangle) \\ &\xrightarrow{\alpha} S_j = (\nu \tilde{q}, d_1, d_2)(D'\langle d_1 \rangle \mid d_1. [\text{return } E]\langle d_2 \rangle \mid d_2. P\langle d \rangle \\ &\quad \mid [\sigma_j] \mid d. \text{Body}\langle \alpha_j \rangle). \end{aligned}$$

Then clearly, $S_2 = (\nu \tilde{u})(Q'_j \mid C') \xrightarrow{\alpha} S'_2 = (\nu \tilde{u})(S'_j \mid C')$ where

$$\begin{aligned} Q'_j &= (\nu \tilde{q}, d_1, d_2, d')(D\langle d_1 \rangle \mid [\text{return } E]\langle d_2 \rangle \mid d_1. d_2. \bar{d}' \mid d'. P\langle d \rangle \\ &\quad \mid [\sigma_j] \mid d. \text{Body}'\langle \alpha_j \rangle) \\ &\xrightarrow{\alpha} S'_j = (\nu \tilde{q}, d_1, d_2, d')(D'\langle d_1 \rangle \mid [\text{return } E]\langle d_2 \rangle \mid d_1. d_2. \bar{d}' \mid d'. P\langle d \rangle \\ &\quad \mid [\sigma_j] \mid d. \text{Body}'\langle \alpha_j \rangle) \end{aligned}$$

and $(S'_1, S'_2) \in \mathcal{R}$.

- Suppose $S_1 = (\nu \tilde{u})(R_l \mid P_i \mid C) \xrightarrow{\tau} S'_1 = (\nu \tilde{u})(F_k \mid G_i \mid C)$ where

$$\begin{aligned} R_l &= (\nu \tilde{r})(E\langle d \rangle \mid [\sigma_k] \mid d. \text{Body}\langle \alpha_k \rangle) \\ &\xrightarrow{\alpha} F_k = (\nu \tilde{r})(E'\langle d \rangle \mid [\sigma_k] \mid d. \text{Body}\langle \alpha_k \rangle) \end{aligned}$$

and

$$\begin{aligned} P_i &= (\nu \tilde{p})(C_i[[S'; \text{return } E]]\langle d \rangle \mid [\sigma_i] \mid d. \text{Body}\langle \alpha_i \rangle) \\ &\xrightarrow{\bar{\alpha}} G_i = (\nu \tilde{p})(S\langle d \rangle \mid [\sigma_i] \mid d. \text{Body}\langle \alpha_i \rangle) \end{aligned}$$

where $C_i[[S'; \text{return } E]] = (\nu \tilde{p}, d_1, d_2)(P\langle d_1 \rangle \mid d_1. Q\langle d_2 \rangle \mid d_2. R\langle d \rangle)$, $P \xrightarrow{\bar{\alpha}} P'$ and

$$S\langle d \rangle = (\nu \tilde{p}, d_1, d_2)(P'\langle d_1 \rangle \mid d_1. [S'; \text{return } E]\langle d_2 \rangle \mid d_2. R\langle d \rangle) = C'[[S'; \text{return } E]]$$

Then $S_2 = (\nu \tilde{u})(R'_l \mid P'_i \mid C')$ where

$$R'_l = (\nu \tilde{r})(E\langle d \rangle \mid [\sigma_k] \mid d. \text{Body}'\langle \alpha_k \rangle)$$

and

$$P'_i = (\nu \tilde{p}, d_1, d_2)(C_i[[S']]\langle d_1 \rangle \mid [\text{return } E]\langle d_2 \rangle \mid d_1. d_2. \bar{d}'\langle d \rangle \mid [\sigma_i] \mid d. \text{Body}'\langle \alpha_i \rangle)$$

Clearly, $S_2 \xrightarrow{\tau} S'_2 = (\nu \tilde{u})(F'_k \mid G'_i \mid C')$, where

$$R'_i \xrightarrow{\alpha} F'_k = (\nu \tilde{r})(E'\langle d \rangle \mid [\sigma_k] \mid d. \text{Body}'\langle \alpha_k \rangle)$$

and

$$P'_i \xrightarrow{\bar{\alpha}} G'_i = (\nu \tilde{p} d_1 d_2)(C'[[S']\langle d_1 \rangle \mid [\text{return } E]\langle d_2 \rangle \mid d_1. d_2. \bar{d}']\langle d \rangle \mid [\sigma_i] \mid d. \text{Body}'\langle \alpha_i \rangle)$$

By construction $(S'_1, S'_2) \in \mathcal{R}$ as required.

To prove the second property suppose $S_1 \downarrow R^-$. First, consider the case where there is no $r : R$ occurring free in S_1 . Then by Lemma 6.6.8, $S_1 \Rightarrow S'_1$ where each object component of S'_1 is in the quiescent state. By the previous property, $S_2 \Rightarrow S'_2$ where $(S'_1, S'_2) \in \mathcal{R}$. It is easy to see by the construction of \mathcal{R} that each component of S'_2 must be in the quiescent state. This implies that $S'_2 \downarrow$ and hence by full Π -confluence $S_2 \downarrow$. Moreover, since $\text{fn}(S_1) = \text{fn}(S_2)$, there are no names of sort R occurring free in S_2 and $S_2 \downarrow R^-$.

So suppose there exists $r : R$ that occurs free in S_1 . Since $S_1 \downarrow R^-$, by Lemma 6.6.8, $S_1 \xRightarrow{\bar{r}(v)} S'_1$, where each object component of S'_1 is in the quiescent state. By the previous property we have that $S_2 \xRightarrow{\bar{r}(v)} S'_2$ where each object-component of S'_2 is in the quiescent state. Thus, $S'_2 \downarrow$, and by full Π -confluence $S_2 \downarrow \bar{r}(v)$. Suppose $S_2 \uparrow \rho$, $\rho \in R^-$. Since S_2 is a derivative of H^b , r is the only name of sort R occurring free in S_2 and $\text{subj}(\rho) = r$. By Π -confluence this implies that $S'_2 \uparrow \rho$. However, $r \notin \text{fn}(S'_2)$ which is a contradiction. Thus $S_2 \downarrow R^-$ as required.

To prove the converse suppose $S_1 \uparrow R^-$. By Property 6.6.20(1), S_2 can mimic any finite or infinite computation of S_1 and thus $S_2 \uparrow R^-$ as required. \square

Let \mathcal{B} be the relation $\mathcal{R} \simeq_1$. We claim that $\mathcal{B} \subseteq \simeq_1^{R^-}$. Suppose $(S_1, S_2) \in \mathcal{B}$ and let S be such that $(S_1, S) \in \mathcal{R}$ and $S \simeq_1 S_2$. Suppose $S_1 \downarrow R^-$. Then by Property 6.6.20(2), $S \downarrow R^-$ and by definition of \simeq_1 , $S_2 \downarrow R^-$. Similarly, if $S_2 \downarrow R^-$ then $S \downarrow R^-$ and $S_1 \downarrow R^-$. Suppose additionally that $S_1 \xrightarrow{\alpha} S'_1$. Then by Property 6.6.20(1), $S \xrightarrow{\alpha} S'$ where $(S'_1, S') \in \mathcal{R}$. There are two possibilities. If $S \downarrow \alpha$ then by definition of \simeq_1 , $S_2 \downarrow \alpha$ and $S_2 \Rightarrow S''_2 \xrightarrow{\alpha} S'_2$ where $S' \simeq_1 S'_2$ and $(S'_1, S'_2) \in \mathcal{B}$. On the other hand, if $S \uparrow \alpha$ then by full Π -confluence $S' \uparrow$ and by \simeq_1 , $S_2 \uparrow \alpha$. Since also $S_2 \downarrow R^-$ it must be that $S_2 \downarrow \Rightarrow S''_2 \xrightarrow{\alpha} S'_2 \uparrow$ where since $S' \uparrow$, $S' \simeq_1 S'_2$. Thus $(S'_1, S'_2) \in \mathcal{B}$.

So suppose $S_2 \downarrow R^-$ and $S_2 \xrightarrow{\alpha} S'_2$. Several possibilities exist. We consider the case $S \downarrow \alpha$. The other case uses similar ideas.

- $\alpha = \tau$. By Π -confluence $S_2 \simeq_1 S'_2$ and hence $(S_1, S'_2) \in \mathcal{R}$;

- $\alpha = (\nu\gamma)\overline{n_G}\langle\gamma\rangle$. Since $S_2 \simeq_1 S$, $S \Rightarrow \xrightarrow{\alpha} Q \simeq_1 S'_2$. This implies that $S = (\nu\tilde{p})(C \mid (\nu\gamma)\overline{n_G}\langle\gamma\rangle.P)$ for some P and C . Moreover, by the construction of \mathcal{R} , $S_1 \xrightarrow{\alpha} S'_1$ so by Property 6.6.16(1), $S \xrightarrow{\alpha} S'$ where $(S'_1, S') \in \mathcal{R}$. By full Π -confluence $S' \simeq_1 Q$, and clearly, $(S'_1, S'_2) \in \mathcal{R}$.
- $\alpha = a * m(\tilde{x}, r)$. Since $S_2 \simeq_1 S$, $S \Rightarrow \xrightarrow{\alpha} Q \simeq_1 S'_2$. Moreover, since S is a derivative of H^b , there exist no names of sort R occurring free in S . By construction, $\text{fn}(S) = \text{fn}(S_1)$, hence no names of sort R occur free in S_1 . Additionally, $S_1 \downarrow R^-$, so by Lemma 6.6.8, $S_1 \Rightarrow S''_1$ where all object-components of S''_1 are in the quiescent state. By Property 6.6.20(1), $S \Rightarrow S'$ where $(S'_1, S') \in \mathcal{R}$ and by construction of \mathcal{R} each object-component of S' is in the quiescent state. Since, by Π -confluence, $S \simeq_1 S'$, $S' \xrightarrow{\alpha} Q' \simeq_1 Q$. This implies that $S' = (\nu\tilde{p})(C \mid \text{Body}(a))$, or $S' = (\nu\tilde{p})(C \mid \text{Body}'(a))$ for some C . So $S''_1 = (\nu\tilde{p})(C' \mid \text{Body}(a))$, for the appropriate C' , and $S''_1 \xrightarrow{\alpha} S'_1$ where $(S'_1, Q') \in \mathcal{R}$. Since $Q' \simeq_1 Q \simeq_1 S'_2$ we have that $(S'_1, S'_2) \in \mathcal{B}$.
- $\alpha = \bar{r}\langle v \rangle$. Since $S_2 \simeq_1 S$, $S \Rightarrow \xrightarrow{\alpha} Q \simeq_1 S'_2$. Moreover, since $\text{fn}(S) = \text{fn}(S_1)$, r occurs free in S_1 and as $S_1 \downarrow R^-$, by Lemma 6.6.8, $S_1 \Rightarrow \xrightarrow{\bar{r}\langle u \rangle} S'_1$. By Property 6.6.20(1), $S \Rightarrow \xrightarrow{\bar{r}\langle u \rangle} Q'$ where $(S'_1, Q') \in \mathcal{R}$ and by Π -confluence, $u = v$ and $Q \simeq_1 Q'$. Therefore, $(S'_1, S'_2) \in \mathcal{B}$ as required.

This completes the proof of the claim. Since $(I^b, H^b) \in \mathcal{B}$, $I^b \simeq_1^{R^-} H^b$. Along similar lines we can also prove that $I^b \simeq_1^{R^-} H^b$. This implies that $I^b \simeq_1^{R^-} I^b$ as required. \square

Chapter 7

Conclusions and Future Work

In this chapter we summarize the results of the thesis and discuss their contributions. We conclude with a discussion of related and future work.

7.1 Conclusions

The need for mathematical frameworks for reasoning about concurrent systems is today widely acknowledged. Understanding these systems requires a clear model, in which intuitive, accessible and precise descriptions can be given, and tools to reason about the model. The purpose of this thesis has been to demonstrate how a variant of the π -calculus can be used to reason rigorously about a class of concurrent systems, systems with dynamically-evolving structure.

Mobility is extensively used in real life. In addition to the applications we have considered, it is common in many systems including operating systems, where the ownership of a resource may change as computation proceeds, telecommunication systems where nodes may move in and out of the communicating range of stationary nodes, and systems supporting process migration which allows tasks to be exchanged to optimize processor usage. At best, these systems can be described in formalisms such as CCS (where mobility is not captured directly) by building a large system where all connections which may be required during computation are established. Drawbacks of this approach are that all potential communications must be known in advance, and it may result in rather awkward and counter-intuitive definitions. For instance, consider the *B*-tree data structure. Reasoning about it in CCS would involve fixing the size of the tree and manipulating a tree of this size throughout computation, where in the initial state all but two of the nodes are idle with the possibility of others being activated as computation proceeds. Simpler descriptions may be given in the presence of parametric channels, or multiway-synchronization

and input-enabledness. Nonetheless, these approaches fail to capture directly the scope migration involved.

Our study has demonstrated that the π_v -calculus, with its small and intuitive set of message-passing primitives, provides a more general and more precise model of concurrency which is especially appropriate for the specification and verification of mobile systems. It can be used to give natural and direct descriptions of mobile concurrent systems which may serve as a basis for their better understanding. Further to that, its formal semantics and mathematical theory enable rigorous analysis of systems' behaviour. An additional benefit of the calculus is that, unlike application-oriented frameworks, it provides a unifying framework for the analysis of different kinds of mobile systems and thus enables concepts and techniques originating in different domains to be generalized and applied in others.

A significant factor for the success of the calculus for the specification and verification of mobile systems will be the development of tools to assist the analysis of π -calculus agents. A number of tools are currently under development. However, because of the complex mathematical theory of the calculus (compared with CCS), standard techniques for checking bisimulations and model checking are not applicable in this context. Among the tools available is the Mobility Workbench [Vic94] whose main functionality is the verification of open bisimulation equivalence ([San93]), a bisimulation strictly finer than observation equivalence. Model checking capabilities are also being developed. Clearly, the analyses we have undertaken in this thesis are outside the scope of existing tools. For one thing, the mobile process calculus we have employed is an extension of the π -calculus with first-order data. Moreover, the correctness criteria for the applications had to be formulated in terms of more demanding equivalences, such as branching bisimulation and indistinguishability within arbitrary contexts. However, improved tools for semi-automatic checking of parts of the analysis would be useful to assist and confirm the correctness of proofs. An interesting question is whether techniques that detect (partially) confluent behaviour of agents can be developed and exploited for automated reasoning.

7.1.1 Determinacy and Confluence in the π_v -calculus

We have investigated the notions of confluence and partial confluence in the π_v -calculus and have developed some of their theory. In particular, we have isolated system-building operators which preserve confluence. This makes it possible to guarantee by construction the confluence of certain systems. The most interesting

of the rules presented states that an *efficient* restricted composition of confluent components is also confluent. This highlights the requirement that for a composition to be confluent it is necessary that its names are used in a disciplined manner by its components. Specifically, it is required that names are uniquely managed throughout computation, that is, a name gives access to at most one resource, and at most one component has access to a resource at any time. We have also provided syntactic conditions which guarantee that these properties are preserved under computation and we have used the results to reason about systems. For instance, observing a single execution of a confluent system allows the detection of the system's capability of producing a certain behaviour, like engaging in a certain action. Furthermore, we have investigated how the confluence of systems can be exploited for systematic verification. This turned out to be especially profitable in the setting of partially confluent client-server systems: in certain client contexts a member of a class of concurrent servers, which may perform up to one state-changing internal action determining the answer to a client's question, is indistinguishable from a serial server.

7.1.2 *B*-tree operations

We have presented a proof of correctness of the concurrent operations on B^{link} -trees. The process calculus provided a direct and natural description of the evolving-structure of the system and enabled a precise statement of its correctness criterion. The analysis highlighted the subtlety of the design and allowed us to formalize the behaviour of the system. Moreover, by forcing consideration of all possible scenarios and computations, it has given grounds for believing that the description is completely defined; a risk to the contrary is often taken during informal system development. It has also clarified nontrivial details and verified assumptions concerning the structure of the tree nodes and the requirement of implicitly storing minimum and maximum keys. Moreover, the proof has provided greater confidence than was afforded by the existing proofs and most importantly it has given valuable insight into why the operations are correct. This insight may prove useful in considering design alternatives and may additionally provide a guide for correctness proofs for a wide range of algorithms for search structures [SG88]. This possibility was illustrated in Chapter 5 by the design of a new deletion operation, a variant of the one proposed in [Sag86], and a sketch of its correctness proof by extension of the existing proof.

An advantage of the type of reasoning undertaken here for concurrency con-

trol is that behavioural equivalence is a correctness criterion applicable to a wide range of systems. By shifting the emphasis to the observable behaviour of a system and ignoring implementation-oriented details it applies in a more general setting. A similar viewpoint was adopted in [LMWF94], where I/O-automata were used for an extensive study of atomic transactions in various types of systems, including locking systems, time-stamped systems and hybrid systems. It appears worthwhile to study the applicability of mobile-process calculi in such settings. The ability to provide natural and clear descriptions of these highly dynamic systems and the powerful techniques for showing behavioural equivalences, seem promising for reasoning about their correctness.

The theory of partial confluence proved to be significant in the correctness proof. It allowed us to perform a number of transformations on the system in question, each step preserving behavioural equivalence, and thus to restrict attention to a simpler system exhibiting a part of the original behaviour. In fact, the results provided by the social confluence theory were sufficient to conclude that the operations in question are serializable. The remainder of the analysis was concerned with proving that each individual operation actually behaves correctly.

7.1.3 Determinacy in $\pi o\beta\lambda$

We have isolated conditions of programs of a concurrent-object language which guarantee determinacy. Although we have worked with a specific language, results similar to those we have presented can be obtained for other languages.

It is not difficult to construct examples to show that if the conditions in the definition of a 'D-program' were omitted, the resulting notion would no longer guarantee determinacy. However, there are of course many programs which although not D-programs are determinate. The nature of the kind of language we consider is such that we cannot hope in general to determine on the basis of a simple syntactic analysis whether a program is determinate or not. In fact, this problem is semi-decidable, that is, it is possible to determine whether a program is not determinate but not to guarantee its determinacy. Nonetheless, we believe that the definition of a 'D-program' may capture a useful programming discipline. Among the benefits of our study may be simpler accounts of determinate fragments of concurrent languages, and improved techniques for the construction of reliable concurrent programs. For instance, the use of syntactic rules which guarantee determinacy provides insurance that a program will exhibit the same behaviour on different implementations of the programming language. This is because by observing a single computation of a con-

fluent program, one may deduce properties of all its possible execution paths. So if an error is present in an instance of the program it will be discovered by running it just once. On the other hand, it is much more complex to reason about a non-confluent program. In particular, it may be necessary to explore all of its execution paths before correctness can be established.

7.1.4 The transformations

The problem of the correctness of transformations of the kinds considered here was posed by C B Jones and has been studied by him in a number of papers [Jon93a, Jon93b, Jon94, Jon96] within a broad program concerned with formal development of concurrent programs. In [Jon93c, Jon93b], the author used the π -calculus to argue about a specific example of the transformations whereas in [HJ95] the correctness of the general transformations was discussed using an operational semantics.

The rules studied in this thesis differ from those cited above in that they enunciate syntactic conditions under which the transformation may be applied. In [Jon96, HJ95] termination of method invocations is a hypothesis of the rules. Here we use a notion of equivalence sensitive to divergence and thereby treat explicitly diverging computations. Further, we exclude the possibility of deadlock by analysing in detail how ‘cycles’ can be created and impose a syntactic condition to prevent this. In [Jon96, HJ95] the transformations refer to a class of program variables called ‘unique references’. According to the definition,

A unique reference is one which is never ‘copied’ nor which has general (unshared) references passed over it.

The conditions we enunciate rely instead on the property that references to a society’s objects are not shared during computation. In contrast to unique references, objects of a society may pass references to non-society objects and also references to society objects. A destructive-read expression similar to the one we consider was also used in [Hog91] concerned with aliasing in a sequential object-language, in which the word ‘island’ was used with a connotation somewhat similar to that of a ‘society’.

A question arising is whether an operational semantics could also be used as a basis for the proof of correctness of the transformations. As mentioned earlier, this method of semantic definition was employed in [PW95] for proving the determinacy of *D*-programs. In fact the proof of the result in that setting was more straightforward than the process calculus proof presented in this thesis. By featuring a less fine level of granularity and a two-level transition system within which to reason

about systems, the operational semantics made it easier to observe the movement of object names in the evolution of the encoding of a program. However, this approach does not appear to be as promising for the correctness proof of the transformation rules. The emphasis now shifted to behaviour, operational semantics which have been proposed in the literature need to be reformulated in order to be employed to reason about a set of classes (a society) occurring within a program context. On the other hand, thinking of objects, and thus sets of objects, as processes provides a natural basis for the type of reasoning required.

7.2 Future work

7.2.1 Determinacy and Confluence

Determinacy and confluence appear to be useful and often desirable properties of concurrent systems. For some systems, establishing trace equivalence as opposed to observation equivalence can be profitable and the ability to restrict attention to only a fragment of a complex system while deducing properties regarding its remaining behaviour can significantly facilitate its verification. Furthermore, the properties capture a sufficient notion of predictability which is required by a variety of systems, and can provide greater confidence against design errors, such as introducing unintended deadlocks. Further study of confluence appears to be worthwhile, and in particular the following two directions seem promising.

Confluence by construction

The confluence-preserving constructions we have isolated are useful for ensuring a system's confluence simply by observing its structure. It is easy to see that if any of the rules were relaxed, for example the conditions imposed on efficient systems, then the resulting rules would no longer guarantee confluence. However, there are many processes which are confluent although they do not fall into these categories. The main aspect of the π -calculus which complicates the treatment of confluence, and its mathematical theory in general, is *name instantiation*. Specifically, on reasoning about an input-prefix it is necessary to consider all inputs which may be received. The notion of sorting can, to some extent, be used to impose some restriction on the kind of entities that may be received in an input. Such additional information turned out to be useful in the design rules we have considered. It would be interesting to explore further the possibilities of a compositional approach to confluence in the π -calculus setting by considering more advanced sorting disciplines, or by isolating

fragments of the calculus in which a simpler treatment of name instantiation and thus of determinacy is possible.

One such arena is offered by the π I-calculus, introduced by Sangiorgi [San95]. This is a name-passing calculus obtained from the π -calculus by restricting communication so that only fresh, as opposed to arbitrary, names can be exchanged between processes (*internal mobility*). Thus a nice property achieved is the symmetry between input and output constructs. Study of the π I-calculus has shown that internal mobility is responsible for much of the expressiveness of the π -calculus (a fully-abstract encoding from an asynchronous variant of the π -calculus to π I has been provided [Bor96]), while it enjoys a much more straightforward algebraic theory. Indeed, it also appears that π I may also allow a simple compositional method for constructing confluent processes. It is easy to see that side-conditions of the rules in Proposition 3.5.1 can be omitted in this setting. For example, consider the rule for input prefix. In the π I we have:

If P is confluent then $a(\tilde{x}).P$ is confluent.

Since the performance of the input $a(\tilde{x})$ will involve only fresh names, there will be no interference between the names received and names already existing free in P . Thus we may prove that confluence is preserved.

Typing Confluence

A central requirement for a parallel composition of agents to be confluent is that its names are used in a disciplined manner. Specifically, it is necessary that the ability to use a name, either for sending or for receiving, is not shared between subcomponents at any time during computation. An exception to this concerns the use of *replicated channels*. In a subcomponent of the form $!(\nu \tilde{y})\bar{a}(\tilde{x}).P$ (or $!a(\tilde{x}).P$), under certain circumstances it is sufficient to require that name a is uniquely handled (or uniquely borne) but make no demand on how it is borne (or handled). In order to guarantee this no-sharing property we have restricted attention to a certain subclass of π_v -calculus processes whose syntax satisfies a number of conditions: (i) the ability to bear a name (to use it for input) cannot be acquired and (ii) the transmission of a name results in losing the capability to use it in non-input position. The type of the conditions involved strongly suggests that by introducing a refined type system for the calculus one might achieve the same effect.

Indeed, a type discipline may not only be used to distinguish the kind of entities which may be communicated via a name but also it may carry additional information about how a name may be used. This possibility was first explored in the

context of the π -calculus in [PS93], where the notion of *polarity* (or directionality) is captured within a type system. This was achieved by introducing two additional link types each specifying that a link can be used only for reading or only for writing. As was illustrated in the same paper this refined notion of sorting yields more powerful techniques for reasoning about systems and, in particular, reduces the number of legal contexts for a given process, thus giving rise to a weaker notion of observation equivalence.

Another dimension of typing processes explored in the literature is that of multiplicity where the type of a channel determines how many times it can be used. One such type system concerned with the case where names can be used linearly (at most once) was presented in [KPT96] for the π -calculus. A similar effect was achieved in [Nes96] where using solely the notion of polarity, type systems were developed in the contexts of CCS and a mobile process calculus where typability of a process guarantees that it uses all of its names ‘uniquely’, that is, at most one component may hold each end of a channel at any time. These kind of type systems provide additional information for reasoning about processes in that in certain contexts (in the absence of the choice operator) two processes communicating over a linear or a unique channel can neither interfere nor be affected by their environment. Consequently, rather strong notions of confluence in the case of [Nes96] and partial confluence with respect to linear types in the case of [KPT96] are satisfied by well-typed processes. Application of a version of the linear sorting of [KPT96] could be exploited in Chapters 5 and 6 and make redundant the verification of A-confluence and R-confluence of systems in Theorem 5.2.13, and Lemmas 6.6.10, and 6.6.11. For example, consider names of sort R in the context of Chapter 6. These are return links via which results of method invocations may be returned. A property satisfied by these names is that they are used at most once: although they may be passed in communications between objects (as the result of method delegation), they may be used at most once for returning the result to the object ultimately responsible for the call. Thus R-names might be assigned linear types, guaranteeing that the agents in question are R-confluent. However, the classes captured by the well-typed processes of these type systems do not coincide with our efficient confluent agents. For one thing, they do not imply the weak confluence notion required by our applications. Nonetheless, these advances are in our view very promising and deserve further investigation. In particular, we hope that the intuition gathered from the study of confluence may lead to a more general type system which characterizes the subclass of confluent efficient agents.

7.2.2 Partial Confluence and Concurrency Control

As observed earlier the form of interaction captured by (Q^+, A^-) -server and (Q^-, A^+) -client agents is very common in computer systems, for example database and memory systems. In addition, the main result we have presented concerning such agents appears to be very closely related to the correctness criteria often associated with these systems, such as serializability and sequential consistency: within a client context a server is indistinguishable from its serial version, that is, the system supporting the same operations but capable of processing at most one operation at a time. The servers studied may perform at most one state changing silent action in order to determine the result of an operation. A great variety of algorithms proposed for dynamic search structures fall into this category and thus our results can be applied fruitfully to establish their correctness. Furthermore, we hope that, by formalizing conditions that are necessary for these type of results, our study may be useful in improving the understanding of systems and algorithms and thus lead to the design of new more efficient variants.

Another challenging possibility for further research is to study how the social confluence theory can be extended to encompass a wider range of systems. In particular, it would be worthwhile to investigate how the definitions of (Q^-, A^+) -client and (Q^+, A^-) -server can be relaxed to allow verification of systems where the server may perform several decisive silent actions before returning the result of an operation invocation. Such results would make it possible to reason and understand more advanced and complex notions of correctness criteria, for example the notion of sequential consistency proposed in [ABM93] and studied in [ABM93, JPZ94, Bri96]. It is not obvious how the results may be extended in this direction though it certainly appears that adoption of a notion of action independence would be useful.

Other promising directions for further study of the notion of partial confluence include connections with non-interleaving semantics and action/process refinement as in e.g. [JPZ91].

7.2.3 Concurrency by transformation

The application of transformation rules for introducing concurrency into sequential programs is a widely studied subject, see for example [Lip75, XH91], which promises a safe mechanism for developing correct concurrent programs. In order to apply transformations safely it is necessary to guarantee that program components do not interfere with each other. In essence the soundness of transformation rules relies on

the fact that given commands S_1 and S_2 , then for all legal contexts $C[\cdot]$,

$$C[[S_1; S_2]] \simeq C[[S_1] \mid [S_2]] \simeq C[[S_2; S_1]].$$

As has become apparent in our study, the notion of confluence is closely related to such correctness criteria. For if $[S_1] \mid [S_2]$ is a confluent agent then there is substantial evidence suggesting that under certain circumstances, equivalences such as the above hold. Thus, it is interesting to see whether the techniques we have employed in this thesis and in particular the notion of confluence can be employed to facilitate program development by transformation. It is also worthwhile to continue the investigation of formal techniques that allow us to prove such results.

Recall that in the context of the transformation rules we have considered in Chapter 6, the sequential and transformed concurrent program fragments have different observable behaviours and thus cannot be compared by the traditional notions of observational equivalence of process calculi. In order to deal with this fact, the criterion of correctness of the transformation rules, formalized in [LW95a] and employed in this thesis, is in terms of the indistinguishability of program fragments within arbitrary program contexts. An alternative approach has been recently proposed by Sangiorgi in [San97], where the correctness of an instance of the transformation was established in the context of a typed π -calculus. The type system employed has the effect of reducing the number of legal contexts for a given process, thus giving rise to a weaker notion of observation equivalence. In particular, the equivalence obtained characterized a sequential and a parallel version of a class as indistinguishable from each other for all possible contexts permitted by the type system. The discipline used was based on the notion of *uniform receptiveness*. It is not clear how this approach can be used to prove a general version of the transformation rules. Nonetheless, in our view, it is worthwhile to investigate this possibility and to compare the applicability of the two approaches.

7.2.4 Concurrent Objects

The desire to exploit the benefits of object-oriented programming in multiprocessor environments has led to much research in the area of concurrent object-oriented programming. Various languages have been designed as a result (e.g. POOL[ABKR86], concurrent C++[Car95], Eiffel[Car93]). However, the effective integration of concurrency and object-orientation still remains a challenge. Considerable effort has been focused on the development of theoretical frameworks for reasoning about concurrent object languages and analysing their constructs. Process calculi in particular seem to be a promising arena for such experimentation.

Various semantics have been provided for object-oriented languages by translation to mobile process calculi and it has become apparent that the notions of name and mobility present in the calculi make them particularly attractive for such purposes, as they capture successfully the basic object-oriented features while offering a basis for reasoning rigorously about them. There are a number of possible directions for further work. First, it is interesting to investigate how the π -calculus can be used more richly to give semantics to typed object languages. To achieve this it is necessary to explore the semantic relation between π -calculus and programming language types. This issue, addressed in [Tur95] in the context of imperative languages, is especially important in object-oriented languages where advanced notions of types and subtypes play a central rôle. Thus the challenge is to investigate the use of type disciplines for the π -calculus which can be used to better understand and reason about typing issues such as subtyping and inheritance. Some progress in this direction is reported in [San96] where a sequential object calculus of [AC94b] is interpreted in a typed π -calculus on the basis of which the subtyping relation of the object calculus is validated. Other issues which may be investigated include communication and synchronization mechanisms and object protection.

Bibliography

- [ABKR86] P. America, J. de Bakker, J. Kok, and J. Rutten. Operational semantics of a parallel object-oriented language. In *Conference Record of the 13th Symposium on Principles of Programming Languages*, pages 194–208. ACM Press, 1986.
- [ABKR89] P. America, J. de Bakker, J. Kok, and J. Rutten. Denotational semantics of a parallel object-oriented language. *Information and Computation*, 83:152–205, 1989.
- [ABM93] Y. Afek, G. Brown, and M. Merritt. Lazy caching. *ACM Transactions on Programming Languages and Systems*, 15(1):182–205, 1993.
- [Abr87] S. Abramsky. Observation equivalence as a testing equivalence. *Theoretical Computer Science*, 53:225–241, 1987.
- [AC94a] M. Abadi and L. Cardelli. A theory of primitive objects: Second-order systems. In *Proceedings of ESOP'94*, volume 788 of *Lecture Notes in Computer Science*. Springer, 1994.
- [AC94b] M. Abadi and L. Cardelli. A theory of primitive objects: Untyped and first-order systems. In *Proceedings of TACS'94*, volume 789 of *Lecture Notes in Computer Science*. Springer, 1994.
- [AC96] M. Abadi and L. Cardelli. *A Theory of Objects*. Springer, 1996.
- [Agh86] G. Agha. *Actors: a model of concurrent computation in distributed systems*. Cambridge, 1986.
- [Ama93] R. Amadio. On the reduction of CHOCS bisimulation to π -calculus bisimulation. In *Proceedings of CONCUR'93*, volume 715 of *Lecture Notes in Computer Science*, pages 112–126. Springer, 1993.
- [Ame89] P. America. Issues in the design of a parallel object-oriented language. *Formal Aspects of Computing*, 1:366–411, 1989.

- [AR87] E. Astesiano and G. Reggio. SMoLCS-driven concurrent calculi. In *Proceedings of TAPSOFT'87*, volume 249 of *Lecture Notes in Computer Science*, pages 45–64. Springer, 1987.
- [AZ84] E. Astesiano and E. Zucca. Parametric channels via label expressions in CCS. *Theoretical Computer Science*, 33:45–64, 1984.
- [BHG87] P. Bernstein, V. Hadjilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.
- [BM72] R. Bayer and E. McCreight. Organisation and maintenance of large ordered indexes. *Acta Informatica*, 1:173–189, 1972.
- [Boe91] F. de Boer. *Reasoning about dynamically evolving process structures*. PhD thesis, Free University of Amsterdam, 1991.
- [Bor96] M. Boreale. On the expressiveness of internal mobility in name-passing calculi. In *Proceedings of CONCUR'96*, volume 1119 of *Lecture Notes in Computer Science*, pages 131–146. Springer, 1996.
- [Bou85] G. Boudol. Notes on algebraic calculi of processes. In *Logics and Models of Concurrent Systems*. NATO-ISA series, F13, 1985.
- [Bou89] G. Boudol. Towards a lambda-calculus for concurrent and communicating systems. In *Proceedings of TAPSOFT'89*, volume 351 of *Lecture Notes in Computer Science*, pages 149–161. Springer, 1989.
- [Bri96] E. Brinksma. Cache consistency by design. *submitted for publication*, 1996.
- [BS77] R. Bayer and M. Schkolnick. Concurrency of operations on B-trees. *Acta Informatica*, 9:1–21, 1977.
- [BW90] J. Baeten and W. P. Weijland. *Process Algebra*. Cambridge, 1990.
- [Car93] D. Caromel. Towards a method of concurrent object-oriented programming. *Communications of the ACM*, 36(9), 1993.
- [Car95] L. Cardelli. A language with distributed scope. *Computing Systems*, 8(1):27–59, 1995.
- [Cas81] M. Casanova. *The Concurrency Control Problem for Database Systems*. Springer, 1981.

- [Cli81] W. D. Clinger. Foundation of actor semantics. Technical report, MIT, Artificial Intelligence Laboratory, 1981.
- [CR36] A. Church and J. B. Rosser. Some properties of conversion. *Transactions of the AMS*, 39:472-482, 1936.
- [Ell80] C. Ellis. Concurrent search and insertions in 2 - 3 trees. *Acta Informatica*, 14(1):63-86, 1980.
- [Ell87] C. Ellis. Concurrency in linear hashing. *ACM Transactions on Database Systems*, 12:195-217, 1987.
- [EN86] U. Engberg and M. Nielsen. A calculus of communicating systems with label passing. Technical report, University of Aarhus, 1986.
- [Eng85] J. Engelfriet. Determinacy \rightarrow (observation equivalence = trace equivalence). *Theoretical Computer Science*, 36(1):21-25, 1985.
- [Gla93] R. van Glabbeek. The linear time - branching spectrum II. In *Proceedings of CONCUR'93*, volume 715 of *Lecture Notes in Computer Science*, pages 66-80. Springer, 1993.
- [GM84] H. Garcia-Molina. Using semantic knowledge for transaction processing in a distributed database. *ACM Transactions on Database Systems*, 2:186-213, 1984.
- [GS95] J. F. Groote and M. Sellink. Confluence for process verification. In *Proceedings of CONCUR'95*, volume 962 of *Lecture Notes in Computer Science*, pages 204-218. Springer, 1995.
- [GW89] R. van Glabbeek and P. Weijland. Branching time and abstraction in bisimulation semantics. In *Information Processing '89*, pages 613-618, 1989.
- [Hen88] M. Hennessy. *Algebraic Theory of Processes*. MIT Press, 1988.
- [HJ95] S. Hodges and C. B. Jones. Fixing the semantics of some concurrent object-oriented concepts: SOS and proofs. In *Proceedings of the Schloß Dagstuhl workshop on Object-orientation with Parallelism and Persistence, to appear*, 1995.
- [Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

- [Hog91] J. Hogg. Islands: aliasing protection in object-oriented languages. In *Proceedings of OOPSLA '91*, pages 271–285. ACM Press, 1991.
- [HP80] M. Hennessy and G. Plotkin. A term model for CCS. In *Proceedings of 9th Symposium on Mathematical Foundations of Computer Science*, volume 88 of *Lecture Notes in Computer Science*. Springer, 1980.
- [HP94] G. Holzmann and D. Peled. An improvement in formal verification. In *Proceedings of FORTE'94*, 1994.
- [Hue80] G. Huet. Confluent reductions; abstract properties and application to term rewriting systems. *Journal of the ACM*, 27(4):797–821, 1980.
- [Jon93a] C. B. Jones. Constraining interference in an object-based design method. In *Proceedings of TAPSOFT'93*, volume 668 of *Lecture Notes in Computer Science*, pages 136–150. Springer, 1993.
- [Jon93b] C. B. Jones. A pi-calculus semantics for an object-based design notation. In *Proceedings of CONCUR'93*, volume 715 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 1993.
- [Jon93c] C. B. Jones. Process-algebraic foundations for an object-based design notation. Technical report, University of Manchester, 1993.
- [Jon94] C. B. Jones. Process algebra arguments about an object-based design method. In *Essays in Honour of C. A. R. Hoare*. Prentice-Hall, 1994.
- [Jon96] C. B. Jones. Accommodating interference in the formal design of concurrent object-based programs. *Formal Methods in System Design*, to appear, 1996.
- [JPZ91] W. Janssen, M. Poel, and J. Zwiers. Action systems and action refinement in the development of parallel systems. In *Proceedings of CONCUR'91*, volume 527 of *Lecture Notes in Computer Science*, pages 298–316. Springer, 1991.
- [JPZ94] W. Janssen, M. Poel, and J. Zwiers. The compositional approach to sequential consistency and lazy caching. Technical report, Universiteit Twente, 1994.
- [KL80] H. Kung and P. Lehman. Concurrent manipulation of binary search trees. *ACM Transactions on Database Systems*, 5:354–382, 1980.

- [KP83] H. Kung and C. H. Papadimitriou. An optimality theory of concurrency control for databases. *Acta Informatica*, 19(1):1–11, 1983.
- [KPT96] N. Kobayashi, B. Pierce, and D. Turner. Linearity and the pi-calculus. *Proceedings of the ACM Symposium on Principles of Programming Languages*, 1996.
- [KS85] J. R. Kennaway and M. R. Sleep. *Syntax and Informal Semantics of DyNe, a parallel language*, volume 207 of *Lecture Notes in Computer Science*. Springer, 1985.
- [KW82] Y. S. Kwong and D. Wood. A new method for concurrency in B-trees. *IEEE Transactions on Software Engineering*, SE-8:211–222, 1982.
- [KY94] N. Kobayashi and A. Yonezawa. Type-theoretic foundations for concurrent object-oriented programming. In *Proceedings of OOPSLA '94*, pages 31–45. ACM Press, 1994.
- [Lam86] L. Lamport. On interprocess communication, parts 1 and 2. *Distributed Computing*, 1(1):77–101, 1986.
- [Lar78] P. Larson. Dynamic hashing. *BIT*, 17:184–201, 1978.
- [Lip75] R. Lipton. Reduction: A method of proving properties of parallel programs. *Communications of the ACM*, 12:717–721, 1975.
- [LMWF94] N. Lynch, M. Merritt, W. Weihl, and A. Fekete. *Atomic Transactions*. Morgan Kaufmann, 1994.
- [Lom83] D. Lomet. Bounded index exponential hashing. *ACM Transactions on Database Systems*, 8(2):136–165, 1983.
- [LW95a] X. Liu and D. Walker. Confluence of processes and systems of objects. In *Proceedings of TAPSOFT '95*, volume 915 of *Lecture Notes in Computer Science*, pages 217–231. Springer, 1995.
- [LW95b] X. Liu and D. Walker. A polymorphic type system for the polyadic π -calculus. In *Proceedings of CONCUR '95*, volume 962 of *Lecture Notes in Computer Science*, pages 103–116. Springer, 1995.
- [LW96] X. Liu and D. Walker. Partial confluence of processes and systems of objects. *submitted for publication*, 1996.

- [LY81] P. Lehman and S. B. Yao. Efficient locking for concurrent operations on B-trees. *ACM Transactions on Database Systems*, 6:650-670, 1981.
- [Mil80] R. Milner. *A Calculus of Communicating Systems*. Springer, 1980.
- [Mil81] R. Milner. A modal characterization of observable machine-behaviour. In *Proceedings of CAAP '81*, volume 112 of *Lecture Notes in Computer Science*, 1981.
- [Mil89] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
- [Mil92a] R. Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2:119-141, 1992.
- [Mil92b] R. Milner. The polyadic π -calculus: a tutorial. In *Logic and Algebra of Specification*. Springer, 1992.
- [ML84] U. Manber and R. Ladner. Concurrency control in a dynamic search structure. *ACM Transactions on Database Systems*, 9:439-455, 1984.
- [MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, parts 1 and 2. *Information and Computation*, 100:1-77, 1992.
- [MPW93] R. Milner, J. Parrow, and D. Walker. Modal logics for mobile processes. *Theoretical Computer Science*, 114:149-171, 1993.
- [MS78] R. Miller and L. Snyder. Multiple access to B-trees. In *Information Science and Systems, Baltimore*, 1978.
- [Nes96] U. Nestmann. *On Determinacy and Non-determinacy in Concurrent Programming*. PhD thesis, University of Erlangen, 1996.
- [Nie89] F. Nielson. The typed λ -calculus with first-class processes. In *Proceedings of PARLE'89*, volume 366 of *Lecture Notes in Computer Science*. Springer, 1989.
- [Nie91] O. Nierstrasz. Towards an object calculus. In *Proceedings of ECOOP'91*, volume 612 of *Lecture Notes in Computer Science*, pages 1-20. Springer, 1991.
- [OP92] F. Orava and J. Parrow. An algebraic verification of a mobile network. *Formal Aspects of Computing*, 4:497-543, 1992.

- [Ora94] F. Orava. *On the Formal Analysis of Telecommunication Protocols*. PhD thesis, University of Uppsala, 1994.
- [Pap86] C. Papadimitriou. *The Theory of Database Concurrency Control*. Computer Society Press, 1986.
- [Pap92] M. Papathomas. *Language Design Rationale and Semantic Framework for Concurrent Object-Oriented Programming*. PhD thesis, University of Geneva, 1992.
- [Par81] D. Park. Concurrency and automata on infinite sequences. In *Proceedings of 5th GI Conference*, volume 104, of *Lecture Notes in Computer Science*, pages 167–183, 1981.
- [PS92] J. Parrow and P. Sjodin. Multiway synchronization verified with coupled simulation. In *Proceedings of CONCUR'92*, volume 630 of *Lecture Notes in Computer Science*, pages 518–533. Springer, 1992.
- [PS93] B. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. In *Proceedings of LICS'93*, pages 376–385. Computer Society Press, 1993.
- [PS94] J. Parrow and P. Sjodin. The complete axiomatization of cs-congruence. In *Proceedings of STACS'94*, volume 775 of *Lecture Notes in Computer Science*, pages 557–668. Springer, 1994.
- [PT94] B. Pierce and D. Turner. Concurrent objects in a process calculus. In *Theory and Practice of Parallel Programming, Sendai, Japan*, pages 187–215. Springer, 1994.
- [PW95] A. Philippou and D. Walker. On sharing and determinacy in concurrent systems. In *Proceedings of CONCUR'95*, volume 962 of *Lecture Notes in Computer Science*, pages 456–470. Springer, 1995.
- [Qin91] H. Qin. Efficient verification of determinate processes. In *CONCUR'91*, volume 527 of *Lecture Notes in Computer Science*, pages 471–494. Springer, 1991.
- [Sag86] Y. Sagiv. Concurrent operations on B^* -trees with overtaking. *Journal of Computer and System Sciences*, 33:275–296, 1986.
- [Sam76] B. Samadi. B-trees in a system with multiple users. *Information Processing Letters*, 5:107–112, 1976.

- [San82] M. Sanderson. *Proof Techniques for CCS*. PhD thesis, University of Edinburgh, 1982.
- [San92] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-order and Higher-order Paradigms*. PhD thesis, University of Edinburgh, 1992.
- [San93] D. Sangiorgi. A theory of bisimulation for the π -calculus. In *Proceedings of CONCUR'93*, volume 715 of *Lecture Notes in Computer Science*, pages 127–142. Springer, 1993.
- [San95] D. Sangiorgi. π -calculus, internal mobility and agent-passing calculi. Technical Report 2539, INRIA, 1995.
- [San96] D. Sangiorgi. An interpretation of typed objects into typed π -calculus. Technical Report 3000, INRIA, 1996.
- [San97] D. Sangiorgi. Typed π -calculus at work: a proof of Jones's parallelisation transformation on concurrent objects. In *Presented at FOOL'97*, 1997.
- [Seg83] A. Segall. Distributed network protocols. *IEEE Transactions on Information Theory*, IT-29(2):23–35, 1983.
- [SG88] D. Shasha and N. Goodman. Concurrent search structure algorithms. *ACM Transactions on Database Systems*, 13:53–90, 1988.
- [ST85] P. Spirakis and A. Tuzhilin. A semantics approach to correctness of concurrent transaction executions. In *Conference Record of the 4th ACM SIGACT/SIGMOD Symposium on Principles of Database Systems*, pages 85–95. ACM Press, 1985.
- [Tho90] B. Thomsen. *Calculi for Higher-order Communicating Systems*. PhD thesis, Imperial College, University of London, 1990.
- [Tho93] B. Thomsen. Plain CHOCS: a second generation calculus for higher order processes. *Acta Informatica*, 30(1):1–59, 1993.
- [Tof90] C. Tofts. *Proof Methods and Pragmatics for Parallel Programming*. PhD thesis, University of Edinburgh, 1990.
- [Tur95] D. Turner. *The Polymorphic Pi-calculus: Theory and Implementation*. PhD thesis, University of Edinburgh, 1995.

- [Vaa90] F. Vaandrager. Process algebra semantics of POOL. In *Applications of Process Algebra*. Cambridge, 1990.
- [Vaa91] F. Vaandrager. On the relationship between process algebra and input/output automata. In *Proceedings of LICS'91*, pages 387–398. Springer, 1991.
- [Vaa95] F. Vaandrager. Verification of a distributed summation algorithm. In *Proceedings of CONCUR'95*, volume 962 of *Lecture Notes in Computer Science*, pages 190–203. Springer, 1995.
- [Vas94] V. Vasconcelos. Typed concurrent objects. In *Proceedings of ECOOP'94*, volume 821 of *Lecture Notes in Computer Science*, pages 100–117. Springer, 1994.
- [VH93] V. Vasconcelos and K. Honda. Principal typing schemes in the polyadic π -calculus. In *Proceedings of CONCUR'93*, volume 715 of *Lecture Notes in Computer Science*, pages 524–538. Springer, 1993.
- [Vic94] B. Victor. A verification tool for the polyadic π -calculus. Technical report, University of Uppsala, 1994.
- [Wal90] D. Walker. Disimulation and divergence. *Information and Computation*, 85:202–241, 1990.
- [Wal91] D. Walker. π -calculus semantics for object-oriented programming languages. In *Proceedings of TACS'91*, volume 526 of *Lecture Notes in Computer Science*, pages 532–547. Springer, 1991.
- [Wal94] D. Walker. Algebraic proofs of properties of objects. In *Proceedings of ESOP'94*, volume 788 of *Lecture Notes in Computer Science*, pages 501–516. Springer, 1994.
- [Wal95a] D. Walker. Objects in the π -calculus. *Information and Computation*, 116:253–271, 1995.
- [Wal95b] D. Walker. Process calculus and parallel object-oriented programming languages. In *Parallel Computers: Theory and Practice*, pages 369–390. Computer Society Press, 1995.
- [Wed74] H. Wedekind. On the selection of access paths in a data base system. In *Data Base Management*, volume Proceedings of the IFIP Working Conference on Data Management, pages 385–397. North-Holland, 1974.

- [XH91] Q. Xu and J. He. A theory of state-based parallel programming by refinement. In *Proceedings of the 4th BCS-FACS Refinement Workshop*. Springer, 1991.
- [Yon90] A. Yonezawa. *ABCL: An Object-Oriented Concurrent System*. MIT Press, 1990.

Index

- α -equivalence, 17
- nil, 14
- τ -inert, 50
- abstraction, 16
- action, 22
 - decisive, 94
 - input, 22
 - output, 22
 - silent, 22
- agent, 16
- agent constant, 16
- agent type, 16
- base type, 14
- bear, to, 33
 - uniquely, 34
- bisimulation, 26
 - db^R -bisimulation, 106
 - branching, 28
 - d-bisimulation, 32
 - db-bisimulation, 32
 - strong, 26
 - weak, 26
- bound, 17
- closed
 - \tilde{I} -closed, 35
 - data term, 15
- component type, 14
- confluence, 50
 - $(Q, A)^\delta$ -social, 124
 - Q, A -social, 109
 - Π -confluence, 72
 - \asymp -confluence, 61
 - d-confluence, 63
 - db-confluence, 65
 - full d-confluence, 66
 - local, 50
 - preserving, 54
- consistent, 33
 - Π -consistent, 73
- constant symbol, 14
- control, to, 34
 - uniquely, 34
- convergence, 31
 - full, 32
- convergent core, 63
- coupled simulation, 29
- cs-equivalence, 30
- cw-equivalence, 30
- data term, 15
- determinacy, 41
 - \asymp -determinacy, 46
 - d-determinacy, 63
 - preserving, 43
 - strong, 41
 - weak, 42
- divergence, 31
- effective agent, 37
- excess, 54

- field selector term, 15
- first-order type, 14
- free, 17
- friendly agent, 33
- guarded, 16
- handle, to, 33
 - uniquely, 34
- inertness, 46
 - S -inertness, 110
 - η -inertness, 64
 - τ_{\times} -inertness, 46
- label, 14
- manage, to, 34
 - persistently, 34
- name, 14
 - bound, 17
 - free, 17
- negative position, 16
- object position, 16
- own, to, 17
- partial confluence
 - R -confluence, 93
 - R^l -confluence, 94
 - R^δ -confluence, 119
 - full R^l -confluence, 98
- partition
 - (M^+, R^-) -disciplined, 102
 - (M^+, R^-) -tidy, 102
 - $(M^+, R^-)^\delta$ -disciplined, 122
 - (M^-, R^+) -ready, 102
 - $(M^-, R^+)^\delta$ -ready, 121
 - $(M^-, R^+)^\delta$ -tidy, 121
 - (Q^+, A^-) -base, 110
 - (Q^+, A^-) -server, 110
 - $(Q^+, A^-)^\delta$ -server, 124
 - (Q^-, A^+) -client, 112
 - $(Q^-, A^+)^\delta$ -base, 124
 - $(Q^-, A^+)^\delta$ -client, 125
 - LMN-confluent, 59
 - LMN-sensitive, 59
- polite, R , 105
- positive position, 16
- prefix
 - input, 16
 - internal, 16
 - output, 16
- process expression, 16
- pruning, 75
- record term, 15
- record type, 14
- replicator, 18
- sort, 14
 - label sort, 14
 - name sort, 14
- sorting, 19
 - M, R -sorting, 101
- stable, 29
- structural congruence, 19
- subject position, 16
- substitution, 18
- trace equivalence, 43
- weak coupled simulation, 30
- weight, 49